Mobile Robot Navigation

Andersen, Jens Christian

Publication date:
2007

Document Version
Publisher's PDF, also known as Version of record

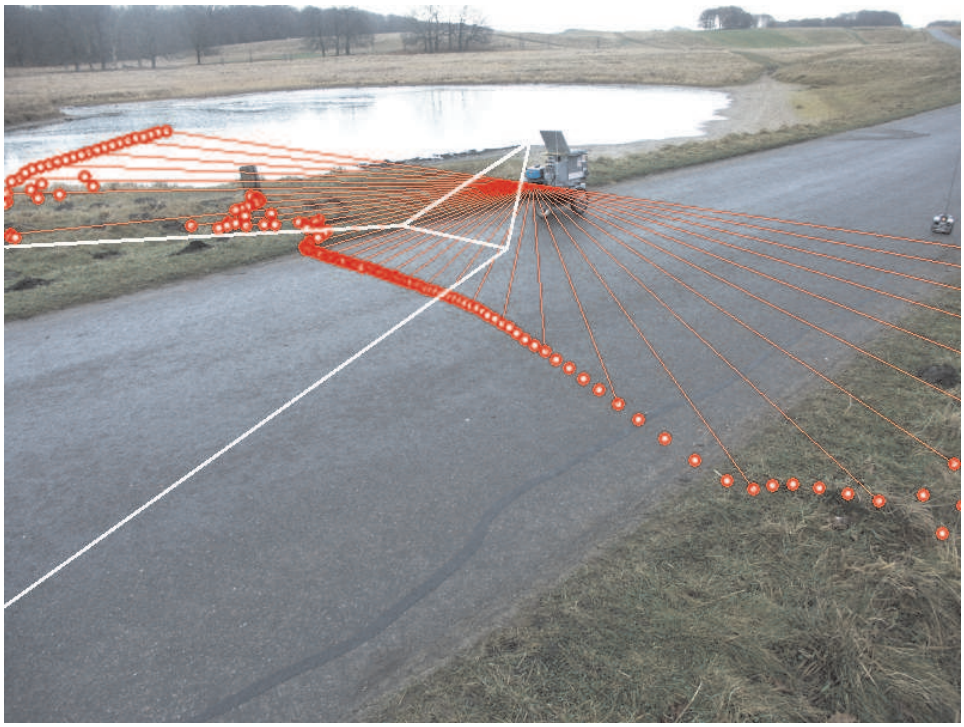
Jens Christian Andersen

Mobile Robot Navigation

PhD thesis, September 2006

Technical University of Denmark

Mobile Robot Navigation



PhD thesis

Jens Christian Andersen

September 2006

Supervisors: Associate Professor Ole Ravn,
Associate Professor Nils A. Andersen
both at Automation Ørsted•DTU

ISBN: 87-91184-64-9

Preface

This thesis is submitted in partial fulfilment of the requirements for the PhD degree at the Technical University of Denmark. The work was performed in the period March 2003–September 2006 at Automation Ørsted•DTU .

The supervisors have been Associate Professor Ole Ravn and Associate Professor Nils A. Andersen both Automation Ørsted•DTU.

Working with robots is an inspiring, satisfactory occupation and a constant source of enjoyment. Especially I remember a small event where a class of about 10-year-old pupils visited our lab. I gave a short demonstration of a few of our robots, and one of the boys asked with wide open eyes: 'Is this a sort of playground for adults?' And when I confirmed with a smile, he replied: 'This is where I want to go when I get older.'

I would like to thank both supervisors for the cooperation and encouragements in the project period and for the gentle pushes – especially in situations where details became prohibitive for progress.

Additionally I would like to thank the other members of the robotics group for the many informal discussions which have been a source of inspiration throughout the years, and to the service-minded workshop for assistance and ideas with the mechanical, electronic and logistic issues when needed.

A special thank to Lisbeth Winter for her (patient) proofreading assistance during the writing of this thesis.

And finally: this project would never have been possible without the support and encouragement from my beloved family: my wife Birte and my son Mikkel.

Kgs. Lyngby, September 2006

J. Christian Andersen

Abstract

Robots will soon take part in everyone's daily life. In industrial production this has been the case for many years, but up to now the use of mobile robots has been limited to a few and isolated applications like lawn mowing, surveillance, agricultural production and military applications. The research is now progressing towards autonomous robots which will be able to assist us in our daily life. One of the enabling technologies is navigation, and navigation is the subject of this thesis.

Navigation of an autonomous robot is concerned with the ability of the robot to direct itself from the current position to a desired destination. This thesis present and experimentally validates solutions for road classification, obstacle avoidance and mission execution. The road classification is based on laser scanner measurements and supported at longer ranges by vision. The road classification is sufficiently sensitive to separate the road from flat roadsides, and to distinguish asphalt roads from gravelled roads. The vision-based road detection uses a combination of chromaticity and edge detection to outline the traversable part of the road based on a laser scanner classified sample area.

The perception of these two sensors are utilised by a path planner to allow a number of drive modes, and especially the ability to follow road edges are investigated.

The navigation mission is controlled by a script language. The navigation script controls route sequencing, junction detection, junction crossing calculations and drive mode selection.

The entire system is tested on a combination of narrow asphalt and gravelled roads connected by a number of junctions. Missions of up to 3 km in length have been successfully completed using the described system.

The main focus of the thesis has been the experimental validation of the implemented solutions and the ability of the methods to solve real world problems.

The amount of software needed by an autonomous robot can be overwhelming. Software reuse and distributed development are therefore important issues. The thesis describes a new component architecture for robotics software that promotes software reuse and distributed development and maintenance.

Dansk Resumè

Robotter vil om få år blive en del af vores daglige liv. Inden for produktionsindustrien har det allerede være tilfældet i mange år, men anvendelsen af mobile robotter har hidtil været henvist til isolerede områder som græsslåning, overvågning, landbrugsproduktion og militære funktioner. Fremskridt i forskningen der gør, at robotter vil kunne assistere os i mange af vore daglige gøremål i en ikke såfjern fremtid. En af de teknologier, der skal gøre dette muligt, er navigation, og navigation er emnet for denne afhandling.

Navigation for autonome robotter handler om robottens evne til autonomt at manøvrere fra den nuværende position til et ønsket bestemmelsessted. Denne afhandling præsenterer og validerer eksperimentelt løsninger til detektering af farbar vej, omgåelse af forhindringer og gennemførelse af missioner. Vejklassifikationen er baseret på laserscanner målinger og assisteret med vision for længere rækkevidde. Vejklassifikationen er tilstrækkelig selektiv til at kunne adskille selv flade vejrabatter fra selve vejen og kan isolere asfaltveje fra grusveje. Vejgenkendelse ud fra kamera billeder tager udgangspunkt i klassifikationen fra laserscanneren og bruger en kombination af farve og kantdetektering til at estimere farbar vej på længere afstande.

Resultatet af disse to sensorer anvendes under planlægning af en farbar rute, og her er det især robottens evne til at følge vejens kanter, der er undersøgt.

Navigationen i en mission er styret af et sekventielt manuskript. Manuskriptsproget giver mulighed for detektering af vejkryds, for beregninger til brug under passagen af disse kryds og til valg a styringsmetode iøvrigt.

Det samlede system er testet på en kombination af asfalt- og grusveje, med et antal forgreninger og vejkryds. Missioner på op til 3 km længde er gennemkørt autonomt med det beskrevne system.

Fokus i afhandlingen har været den eksperimentelle validering af de implementerede metoder og metodernes evne til at løse problemer i en virkelige verden.

Der skal en betydelig mængde software til for at styre en autonom robot, emner som software genbrug og distribueret udvikling er derfor essentielle. Denne afhandling beskriver yderligere en komponentbaseret arkitektur for robotter, som kan fremme software genbrug og distribueret udvikling og vedligeholdelse.

Contents

Chapter 1

Introduction

Navigation is in Encyclopædia Britannica defined as *'science of directing a craft by determining its position, course, and distance travelled. Navigation is concerned with finding the way to the desired destination, avoiding collisions, conserving fuel, and meeting schedules'*. Mobile robot navigation is thus the ability of a mobile robot to get from one place to another in an orderly manner.

For a navigation robot the purpose is to get to a destination, but you do not buy a robot for this purpose alone. To solve real problems the robot must be able to do something, a gardening robot must be able to do some gardening like cutting the hedges, and a delivery robot must be able to deliver items like mail or groceries, a surveillance robot must report on the surveillance findings. Navigation is the common ability that enables the robot to reach the destination required by the job.

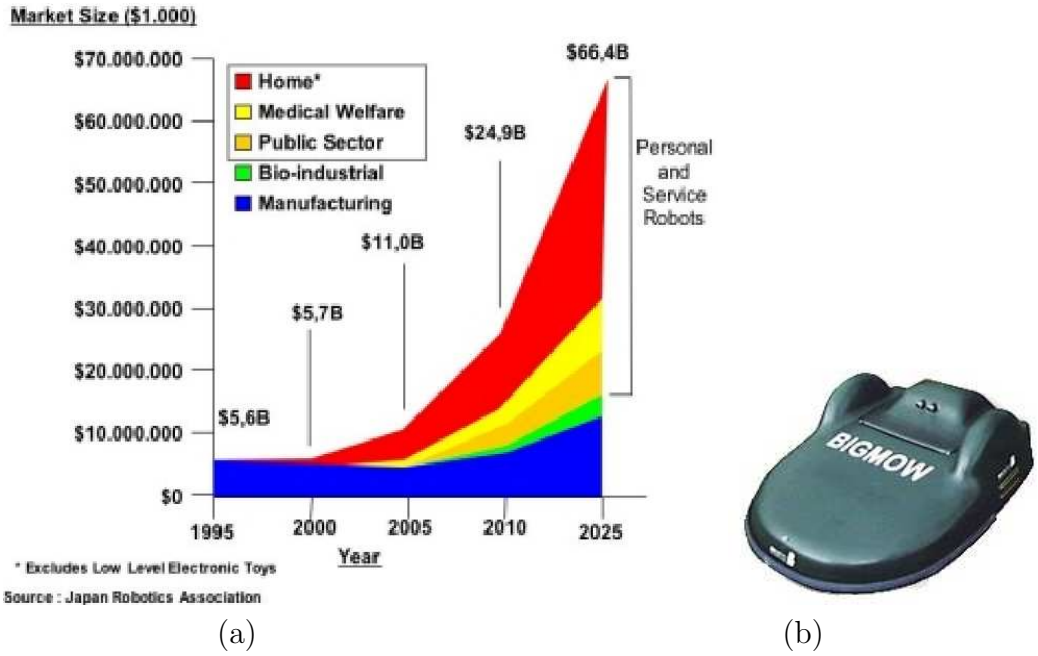
This thesis deals with the enabling technologies needed by a navigating robot. The method is to research methods and test their ability to cope in real world situations.

1.1 State of the art

The UN publishes annually a status report *World Robotics* (2005) on the use of robotics – covering both industrial and service robots – based on contributions from the member countries. The service robot area is small compared to the industrial robots, but it is a fast growing area and is expected to exceed the market size for industrial robots within the next few years. The groth is estimated by the Japan Robotics Association as shown in Fig. 1.1(a).

The European Union has established the European Robotics Platform (EUROP) (Barontini et al. (2005)) to promote the robotics area, and expects Europe to play a major role especially in the area of non-military service robots. At present the USA is dominating the area of military robotics and Japan the area of domestic (entertainment) robots.

Robots for autonomous lawn moving and vacuum cleaning have been available commercially for a number of years; two recent examples of products are shown in Fig. 1.1(b) and Fig. 1.2(b). The solutions have very limited perception of the working area, and the working area is covered using a random rather than a systematic and optimised method.



(a) (b)

Fig. 1.1: The service robot market is expected to exceed the market for industrial robots within the next few years (a). Automatic lawn mower from Belrobotics (b).

Within the agricultural area GPS[1] has made a spectrum of field robotic applications attractive, an example of a robot in the area of crops and plant nursing is shown in Fig. 1.2(a) from the Danish Royal Veterinary and Agricultural University (KVL) and is described in Mejnertsen & Reske-Nielsen (2006).

Robots for storage and recovery of goods in storage systems are already in place, and robots for delivery to and from end users in hospitals are seen at places. In the future such systems could be expanded to deliver mail in office buildings, or to deliver the ordinary mail and other commodities to private houses.

Potentially computer-driven cars could make the traffic on highways and motorways more efficient, some results have been demonstrated by Thorpe et al. (1997), and an image from this event is shown in Fig. 1.3(a), where a number of cars are driven at high speed with short separation.

[1]GPS: Global Positioning System

(a) (b)

Fig. 1.2: Autonomous tractor (a) from the Royal Veterinary and Agricultural University (KVL); an Electrolux robotic cleaner (b).

Surveillance of areas of high value is an area for robots when camera surveillance or guard dogs are either impractical or insufficient. The present technology in the area has a limited autonomy though, like the OFRO robot by Robowatch (2006) shown in Fig. 1.3(b) this was used during the football world cup event 2006 in Germany. Other areas could be monitoring of agricultural crops, inspection of deserted mines, areas with dangerous chemicals (like near an active volcano) or radioactive waste (a depot or after an accident).

Handicap assistance is expected to be a major area for robots in general, but also for autonomous robots, and could help a number of elderly or disabled people to be less dependent on human assistance. One of the future examples in this area could be a robot acting as a guide dog for blind people.

Search and rescue are areas, where robots could save lives without putting the rescue workers at risk. The rescue robots on the accident scene are primarily envisaged as guided robots, but autonomous robots could be beneficial if large areas were to be searched for survivors, like in an avalanche area.

The military applications include ground transportation of goods across potential hostile areas to support friendly units or own troops. One of the projects that aim for this segment is described in Aufrere et al. (2003) where a large number of sensors is used in an attempt to create an autonomous outdoor platform. The USA defence organisation DARPA[2] has sponsored a number of projects in the area and has organised the Grand Challenge 2004 and 2005 events for autonomous vehicles. Five teams completed in 2005 the 132 miles desert terrain route; the top two of the participants are shown in Fig. 1.4. A

[2]DARPA: Defence Advanced Research Projects Agency

(a)

(b)

Fig. 1.3: Demonstration of computer-driven cars travelling at 65 miles per hour in San Diego 1997 (a). Surveillance robot used during the football world cup games in Germany 2006 (b).

number of papers and reports are released from these events, eg Behringer et al. (2005), Xiang & Ozguner (2005), and the winner from Stanford University described in Thrun et al. (2006).

1.2 Unsolved aspects

The EUROP describes in Barontini et al. (2005) cognition as one of the main areas where further research is needed. The cognitive system must base its decisions on situation awareness. And situation awareness requires for a robot a qualified perception of the surroundings. Most of the future robots are expected to operate in an unstructured environment where such skills are needed.

In research work published over the last 20 to 30 years there are suggestions to solutions in almost any of the individual disciplines needed to build an autonomous robot. A plenitude of drive system solutions is available. Sensor solutions based on sonar, laser, vision, radar, telemetry, odometry, inertia and others are described, and each description presents aspects that may be beneficial for an autonomous robot. When entering into the higher levels of abstraction in perception and cognition there are less published results.

The perception needed by an autonomous robot is primarily the positioning of nearby objects and a qualified type identification of the objects. When on a highway with objects in front of the vehicle and on the right side, it is important to know that the object in front is a car and the object on the right is the road edge – and not vice versa.

The question as to the position of the objects is often readily solvable by sensors like a laser scanner, whereas the determination of object types often requires more data processing and often more data. A vision sensor requires

Fig. 1.4: From the DARPA Grand Challenge 2005, the team from Stanford University (a) and from the Carnegie Mellon University (b).

much more processing to determine where the surrounding objects are located, but includes typically other aspects – like colour – of the objects. A number of advances have been published recently in object type determination using vision; an example is Fergus et al. (2003) who is using a feature detection method from Kadir & Brady (2001) to recognise objects like a motorbike, a plane or a panther in a series of images. The method recognises a set of scale related features from the image and from the feature types and their relative position the object type is determined. The method is complex and time consuming, but this and a number of other advances in vision-based perception is promising for the future, as summarised in Kragic (2005).

One of the key areas in researching the more advanced issues of autonomous robots like cognition is the availability of a standardised basic platform. The research for specific solutions used in robotics is typically performed in small groups, and to establish new results, these should preferably be based on previously described results. The previous results are often not available to others than the researcher and possibly to the colleagues at the research establishment. A standardised software architecture could reduce the time needed to recreate already established results, and thus would allow more research time for new areas. Some software component standardisation is already taking place in for example the CARMEN project (Montemerlo et al. (2003)) for robot components in general, and the system described by Ponweiser et al. (2005) for a vision-based system.

The conclusion is that especially within the perception and cognitive disciplines there are many unsolved aspects, and these are the areas that suffer most from the lack of a standardised component framework. A number of such components – like sensor interface, drive control, simulation and mission scheduler – are needed before perception and cognition components can be integrated and tested in a real world situation.

It will still take a number of years before robots are able to navigate autonomously and safely in populated areas.

1.3 Objectives

The research objectives of this thesis are to experimentally verify navigation solutions for mobile robots and their ability to cope with real world situations. The experimental approach is sketched in Fig. 1.5 and is in general used to

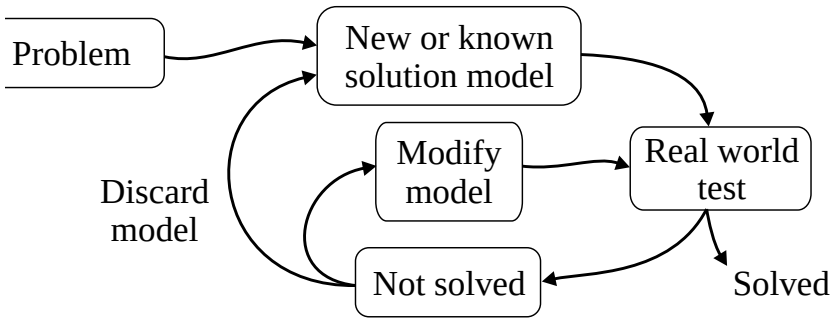


Fig. 1.5: The objectives of this thesis are to take navigation solutions to the test. This experimental robotics approach is used to create new, to modify or discard solution models.

verify models or assumptions through experiments. The results should include a navigating robot and a software architecture that would allow reuse in other projects.

The navigation solutions investigated should be broad in scope, but may be evaluated against a more limited environment. A specific test area has been selected; the area is a nearby nature reserve (Dyrehaven near Copenhagen). The tests should include autonomous navigation on both asphalt and gravel surfaced roads and junctions in the test area.

Obstacles are to be avoided, but moving obstacles need not be handled specifically.

1.4 Navigation disciplines

Navigation is concerned with finding the way to a desired destination. This can be divided into three parts: the localisation, where am I; the mission, where do I want to go; and finally how do I get there.

Where am I

I am at 55.7986°N and 12.5456°E or at 346145.264E 6183922.977N in zone 33 of the UTM[3] projection heading 72.5°, may be an accurate positioning, but a position like 'facing building 326' or 'eastbound on route 20' may be much more useful as navigation reference – dependent on the situation.

The important issue is a knowledge of own position relative to the destination.

Where do I want to go

The mission objectives for a navigating robot are to get to a different place. It could be to a position far away, or a position with a number of specific waypoints, it could be just to explore the surroundings, or search an area until some success criterion is met.

There may be a number of competing missions, eg the main mission may be to explore the area to the left, but the batteries may be almost flat and the charging station is to the right.

These mission types can all be broken down into a sub mission of the type: bring the robot from A to B.

How to get there

An old Chinese proverb says: *'Even the longest journey begins with a first step'*. This could be translated as: get moving in the right direction, avoiding obstacles as they are detected. This reactive approach may work well, and is in the spirit of Allan Brooks (MIT 1987) *'Planning is just a way of avoiding figuring out what to do next'*, but a plan may be needed in some situations, eg when the robot has met a dead end and the final destination is further ahead.

If a map of the area from A to B is available, with traversable terrain information, then it may be possible to make an overall plan to get to B, and possibly to select the most appropriate from a number of alternative routes. But as the obstacle situation along the route most likely is unknown the detailed planning must wait until the obstacle situation is detected.

[3]UTM: Universal Trans Mercator

Thus a solution should combine long term planning – based on past experience, eg a map – and a short term reactive behaviour to cope with detected obstacles and situations not foreseen in the map.

1.5 Navigation platform elements

The elements needed for a navigation robot are (see also Fig. 1.6):

- a mechanical system that is capable of making the move from A to B,
- sensors to get impressions from the current situation,
- cognition to perceive the sensed information and to decide on a behaviour, and
- a control system to implement the desired behaviour.

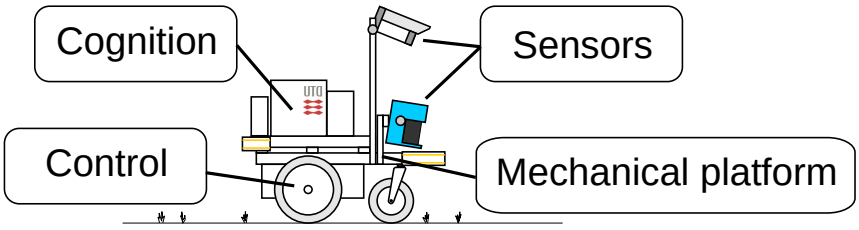


Fig. 1.6: The main elements of a navigating robot are: sensors, a cognitive function, a control system and a mechanical platform.

Cognition is (in this sense) the process of reasoning on the perceived information and from the reasoning the making of intelligent behaviour decisions. Perception is the interpreting and organising of the sensory information.

A car is a mechanical system capable of making the move from A to B, and it has a control system to implement the desired behaviour. The driver handles the remaining elements: the sensors (primarily using the eyes and ears) and the perception of the sensor data (into road and traffic and the interpretation of signposts) as well as the decision of an appropriate behaviour (eg take the next exit and stop when the target position (B) is reached). A normal car is thus not a navigation robot, as the car does not handle the sensing and cognition.

A (modern) cruise missile has all it takes to get from A to B autonomously, it has sensors that allow perception of the current situation, it has sufficient cognition to correct the course as needed and to take the appropriate action when the target is reached. A cruise missile is thus a navigating robot.

1.6 Contributions

This thesis focuses on the autonomous navigation problem with special emphasis on outdoor navigation in the semi-structured environment of the test area.

The main contributions from this thesis are:

- A laser scanner (2D) based perception of road surface and obstacles. This is the main sensor for obstacle avoidance and environment perception. The classification into traversable road surface and nontraversable terrain is in short form described in the papers (Blas et al. 2005) and (Andersen et al. 2006).

- A vision-based perception of the available road in front of the robot. The purpose is to extend the decision range for direction decisions beyond the laser scanner range. The results were presented at the 10th International Symposium on Experimental Robotics 2006 (Andersen et al. (2006)).

- A guidemark solution to detect artificial 2D guidemarks using vision. The solution can evaluate the robot position relative to the guidemark position. The guidemark includes a (unique) code and can thus be used as fixed reference points in the navigation process.

- A behaviour control scheduler, where a sequence of prepared navigation routes can be executed, including decisions based on events on the route, eg detection of guidemark, or calculations based on sensor measurements etc. This is the human interface point for the implemented navigation solution, and is intended as the interface point for a mission planner above the navigation level. The navigation system has proved its value on a 3 km autonomous drive on different road types within the test area.

- A software component architecture dividing the software complexity into a client server structure where the server components complexity is further reduced by moving parts of the functionality to plugins. The client server interface uses an XML[4] type text interface, this allows easy extension, monitoring and debugging.

1.7 Thesis structure

The thesis is divided into an

[4]XML: Extensible Markup Language.

Overview chapter of the used robot solution, and is describing the overall functionality decisions and design.

Following the overview chapter a separate chapter is allocated for each of the main functional areas:

Laser scanner based sensing includes the feature extraction used to separate traversable road areas from nontraversable road sides and other obstacles.

Vision-based feature extraction describes the determination of available road area at longer ranges and the position determination of artificial guidemarks.

Obstacle avoidance includes obstacle management, sensor fusion and determination of the short term path planning in reaction to the detected obstacles.

Mission planning and execution include the navigation scheduler which combines the planned mission with obstacle avoidance.

Following these functional chapters a separate chapter is allocated for a proposal for a new standardised system architecture.

Software architecture is proposing a new division of the robot functionality into separately maintainable components and functional modules.

Each of these chapters includes some test results for the described subsystem. The final two chapters primarily deal with the full system.

Results where the results obtained on the full system are discussed.

Conclusion which is a summary of the results.

The appendices hold sections describing results and detailed information expanding on the results in the main chapters.

Chapter 2

Overview

2.1 Introduction

Outdoor navigation in open terrain requires sensors that can detect obstacles and traversable terrain directly or indirectly, it requires sensors to determine own position relative to the obstacles and relative to desired target position, and finally to implement the navigation decisions, it requires mobility abilities in the desired terrain.

These requirements are discussed in this chapter and are used to determine a set of requirements for an outdoor navigation solution.

Some navigation subsystems are then described in the following chapters and tested against the requirements.

2.2 Robot conceptual model

The National Aeronautics and Space Administration (NASA) and the US National Institute of Standards and Technology (NIST) have developed a Standard Reference Model Telerobot Control System Architecture (NASREM) described in Albus et al. (1986) and extended it for intelligent system design as in Albus (1992). This model has been used as the conceptual framework for many robot projects since then. The basic structure for this model is shown in Fig. 2.1.

The right column in Fig. 2.1 holds the behaviour generation functions, where the overall mission is sequenced into a specific behaviour to achieve the mission goal. The mission assignment is entered at the top to the task scheduler that implements the tasks needed to progress the mission and reflects on the current state of the system. The relevant system state is supplied by the world model, eg the current position on the navigation map.

The obstacle avoidance block takes the task assigned from the task scheduler, eg follow road 200 m forward, and decides on a route that brings the robot

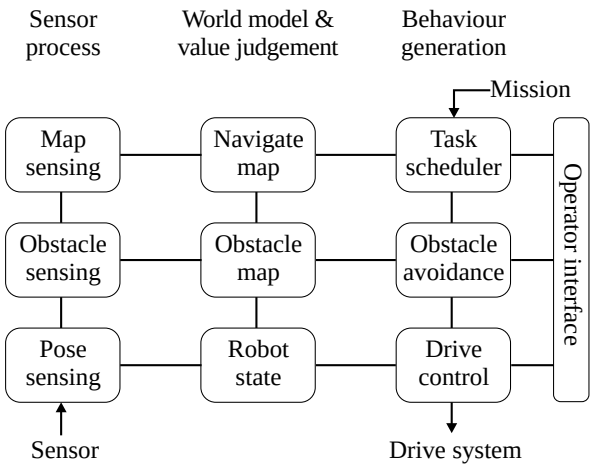


Fig. 2.1: The conceptual architecture for the robot. The sensors with basic data extraction are in the left sensor column, the centre column holds the model of the world and the robot as well as the current state. The right column is the tasks that control the robot behaviour from the more abstract level on top to the detailed motion control at the bottom.

in the right direction considering the current obstacles, terrain and robot state.

The drive control implements the drive command by controlling the individual actuators in the drive system, eg implementing the requested speed and heading.

The sensor process blocks extract the needed data from the real world; the intention is that this information can be used to maintain a model of the world that is sufficient for the behaviour decisions.

The value judgement part of the world model column can evaluate alternative plans in terms of cost, risks and benefits, add uncertainty, attractiveness and desirability to measurements and states in the world model. The result of the value judgement can then be used to improve behaviour decisions.

In the behaviour generation column the planning horizon will decrease as the task decomposition gets closer to the drive system actuators. The task decomposition may be hierarchical, so that the task scheduler may issue navigation tasks to the obstacle avoidance planner and at the same time related tasks to a manipulator or some other subsystem of the robot. The obstacle avoidance task may issue commands to both the drive system and other systems like eg an audible alarm or an obstacle movement system. The drive control – most likely – will control a number of actuators.

2.3 Task breakdown

The task breakdown shown in Fig. 2.2 is an expansion of the conceptual model presented in Fig. 2.1. The experimental platform (described in section 2.4) sets some of the possibilities and limitations, and these are included in parts of the argumentation for the task breakdown.

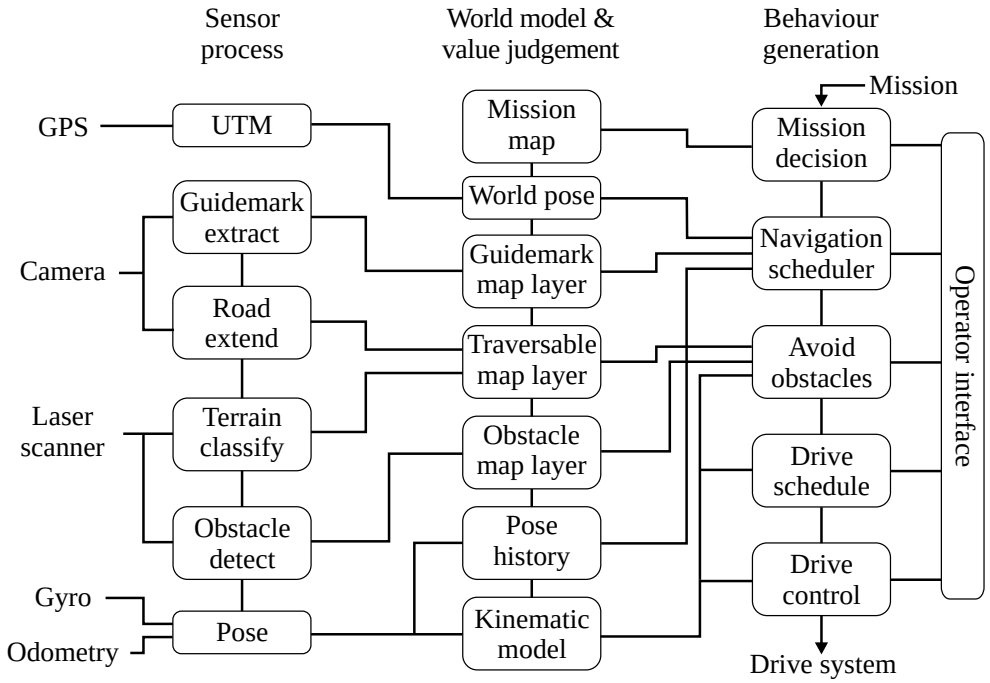


Fig. 2.2: The conceptual architecture expanded to include first level of functional breakdown. The implemented sensors with data extraction are in the left column, the centre column holds current state and the model of the world as generated by the sensors and the planned behaviour. The right column holds the tasks that control the robot behaviour. The mission map and the mission assignment blocks are not implemented. The operator interface is available primarily to initiate and monitor the functionality.

2.3.1 Behaviour generation

Missions are loaded to the ***mission decision*** block and are assigned to the *task scheduler* after some sort of priority scheme. These parts, neither the *mission map* nor the *mission decision* blocks, are implemented or investigated in this thesis. The mission priority list could eg hold: go to position B, C or D, explore new roads around position E or return to docking station. The top

priority mission should then be compiled into a mission sequence of navigation commands – eg covering the route from current position to point C – and present this to the task scheduler.

The implemented navigation command set in the *task scheduler* includes: 'follow road (left, right or centre)', 'goto (relative) position' and a set of conditions that determine when a command is finished (ie a stop criterion). Additionally, any of the behaviour primitives provided by the drive scheduler may be used too. The command set supports relative or topological navigation commands, either relative to the road or relative to the current posture or to the posture history. When following roads and crossing junctions this command set should be sufficient. The only GPS support implemented is as a stop condition, where it can be used to determine when to expect eg a junction or a new road type.

The ***navigation scheduler*** will then in sequence present a method to traverse part of the route – eg keep left on this road or how to cross a square – to the *avoid obstacle* function, and monitor the available progress indications (used as a stop criterion) before continuing to the next part in the sequence. A stop criterion could be: detection of a vision-based guidemark, that the robot is near a given GPS position, that the robot has travelled a given distance or that no route is available. The *task scheduler* will then decide what to do next, according to the assignment. This area is covered in detail in chapter 6.

The ***avoid obstacles*** function will attempt to follow the directions presented by the *task scheduler* by planning a route that follows traversable areas of the terrain and avoids obstacles. The found route is formed as a sequence of drive commands – like 'forward 1m' and then 'turn 20° right' – and presented to the *drive scheduler* replacing the previously assigned drive schedule. Obstacle management and avoidance are the subjects of chapter 5.

The ***drive scheduler*** implements the drive schedule taking the current state and kinematics model of the robot into account, ie aiming for a desired speed at an acceptable acceleration. The result is then presented to the ***drive control*** for the relevant part of the drive system. The interface to the drive scheduler is described in chapter 5.

The *drive scheduler* (named *SMRDEMO*) and the *drive control* are not developed as parts of this thesis.

2.3.2 World model and value judgement

The ***world posture*** is intended to maintain a global reference to the mission map using a GPS. The primary purpose of the *world posture* is to be able to determine if a reference point is reached in the mission map. Aiming directly for a position in GPS coordinates is not implemented, but could be an extension to the allowed command set. For this purpose a standard GPS unit is available

and allows the robot to get a reference position in absolute coordinates.

The ***guidemark*** map layer holds information about detected guidemarks, ie which guidemarks are observed where. A number of guidemark types could be interesting, eg a road sign, a wall, a house, a road or a tree, or other objects that could be used to recognize a given position. Of these guidemark types only roads are implemented, and the available information includes estimated road width and the estimation quality. A vision-based artificial guidemark sensor is available; this detects an artificial two-dimensional barcode (a checkerboard code). The guidemark code as well as its (3D) position and orientation are detected. The guidemark extraction is discussed in chapter 4.

The ***traversable map*** and the ***obstacle map*** are parts of the feature map used for local navigation for obstacle avoidance. Lack of obstacles is not the same as good traversable terrain. The roads in the test area are typically edged by slightly rougher grass areas, some of which may be traversable by the robot, but the robot should preferably stay on the roads. Obstacles, on the other hand, are defined as areas that hinder progress and thus should be avoided at all times. The traversable map can therefore be used in the value judgement for possible alternative routes.

The ***pose history*** holds the newest posture history of the robot. The intention of this history is recognition of road curves and the average road direction. This information is available to the *task scheduler* and may be used to determine when to advance to the next command in the navigation sequence.

The ***kinematics model*** is needed to generate smooth and controlled manoeuvres.

2.3.3 Sensor processing

The ***laser scanner*** is selected as the main sensor for terrain following and obstacle avoidance. As the terrain is primarily flat, the laser scanner is tilted downwards to get measurements from the road and roadsides. The measurements are used for terrain classification and for obstacle detection. This is the subject of chapter 3.

The ***vision sensor*** is used for road detection beyond the reach of the laser scanner and for detection of artificial guidemarks. This issue is discussed in chapter 4.

A ***gyro*** is used to supplement the wheel ***odometry*** sensor to get a more stable heading. The odometry based heading is very sensitive to terrain curvature and structure. The combined wheel odometry and gyro sensor provides stable posture estimate in the tested environment.

2.4 Experimental platform

The robot platform used for the tests was constructed as part of a master's thesis Nielsen & Breiting (2004) in 2004.

2.4.1 Mechanical capabilities

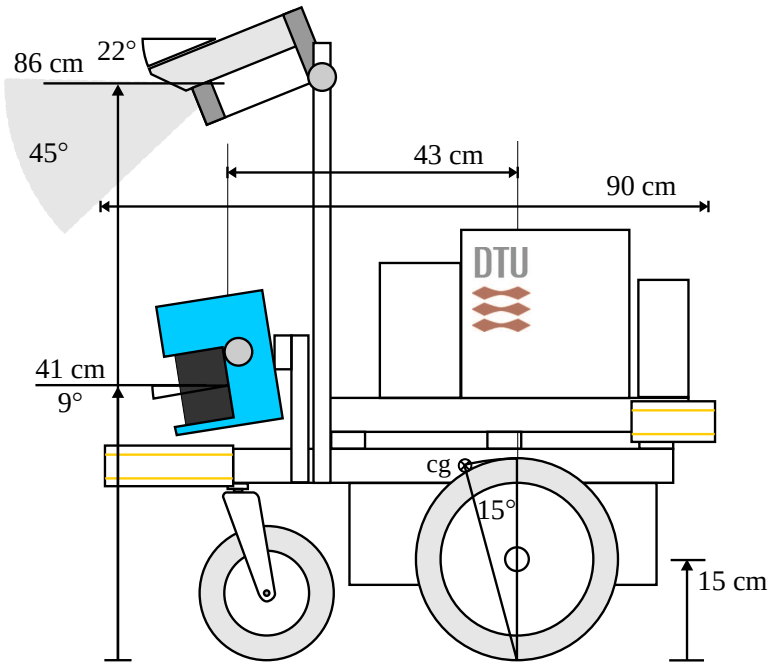


Fig. 2.3: The experimental robot is 90 cm long and 65 cm wide and weighs about 77 kg. The rear wheels are electrically powered, the wheel base is 45 cm and the wheel diameter is 30 cm. The front caster wheel has a 20 cm diameter. The suspension includes rubber shock absorbers. The SICK laser scanner is tilted down by 9° and the camera by 22°. The centre of gravity (cg) is positioned close to the driving wheels.

The robot is built to access the same terrain types as are defined in the Danish standard BR-95 for wheel chair mobility. It is 65 cm wide, 90 cm long as sketched in Fig. 2.3. It weighs 77 kg and has a top velocity of 10 km/h.

The robot is powered by two electrically driven rear wheels, each with a 180 W DC motor. A single caster front wheel ensures static stability. The differential steering uses the rear wheels only.

The motors can each produce a sustained torque of $T = 13.4\,\mathrm{Nm}$ on the wheel, and with a total weight of $M = 77\,\mathrm{kg}$ and a wheel radius of $r = 0.15\,\mathrm{m}$,

the maximum road inclination that the robot can climb is $R_i = 13.6°$ as shown in (2.1).

$$R_i = \sin^{-1}\frac{2T}{rMg} = \sin^{-1}\frac{2 \times 13.4}{0.15 \times 77 \times 9.8} = 13.6° \tag{2.1}$$

This is approximately the same as the angle to the centre of gravity seen from the rear wheels. This indicates that the robot in this configuration will be able to do a wheelie (lifting the front wheel), especially as the motors will be able to deliver a higher peek torque.

To avoid a wheelie the total torque should be limited to $T_w = 31.4\,\mathrm{Nm}$ as shown in (2.2), or 15.3 Nm each wheel. This corresponds to a wheelie acceleration $a_w = 2.7\,\mathrm{ms}^{-2}$ as shown in (2.3)

$$\begin{aligned} Mgx_{cg} &= \frac{T_w}{r}h_{cg} \\ T_w &= \frac{Mgx_{cg}r}{h_{cg}} = \frac{77 \times 9.8 \times 0.15 \times 0.075}{0.27} = 31.4\,\mathrm{Nm}, \end{aligned} \tag{2.2}$$

where $x_{cg} = 0.075\,\mathrm{m}$ is the position of the centre of gravity in front of the rear wheels, and $h_{cg} = 0.27\,\mathrm{m}$ is the height of the centre of gravity, as shown in (2.3)

$$a_w = \frac{T_w}{rM} = \frac{31.4}{0.15 \times 77} = 2.7ms^{-2}. \tag{2.3}$$

The weight distribution is approximately 13 kg on the caster wheel and 32 kg on each of the rear wheels.

The robot is equipped with a 38 Ah 24 V power pack; this should ensure continuous driving for about 7.5 hours at a speed of about $1.5\,\mathrm{ms}^{-1}$. The electronics alone uses approximately 50 W corresponding to about 18 hours of idle operation time.

2.4.2 Posture detector

Wheel encoders on both driving wheels ensure odometry distance measurements for each wheel. The difference in wheel distance can be used to estimate heading change, but this estimate is dependent on a known wheel diameter. The air-filled tires allow for some compression if the load increases, and this will influence the effective wheel radius and hence the odometry accuracy. On a curved road as shown in Fig. 2.4 the roll angle results in an uneven weight distribution as shown in table 2.1.

The wheel depression is measured to approximately $0.0001\,\mathrm{mkg}^{-1}$ (derived from variations in distance driven with different loads) at a tire inflation pressure of (about) 1.5 bar. The difference in wheel radius $(r_l - r_r)$ relative to the

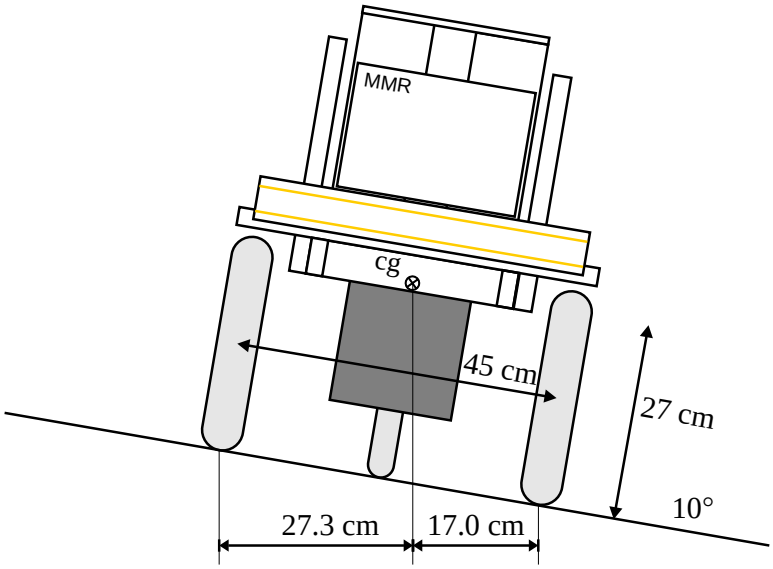


Fig. 2.4: On curved roads the roll of the robot will shift the weight balance resulting in a change in wheel compression and thereby influence the effective wheel radius used in the odometry.

wheel base ($B = 0.45$) is the same as the average wheel radius ($r = 0.15$) relative to the turn radius (R), as shown in (2.4) and exemplified for a roll angle of 10°

$$\begin{aligned} \frac{r_l - r_r}{B} &= \frac{r}{R} \\ R = \frac{Br}{r_l - r_r} &= \frac{0.45 \times 0.15}{0.15065 - 0.14935} = 52 \text{ m}. \end{aligned} \tag{2.4}$$

In outdoor terrain a roll angle of 2–5° on either side of an asphalt or gravelled

Table 2.1: The effective wheel radius of the wheels depends on the tyre depression due to load. When the road profile is curved and the wheels are turning at the same rotation speed, the robot will follow a curve with the shown turn radius, or suffer the shown systematic heading error (tyre depression effect only).

Roll	Load [kg]		Radius [cm]		Turn	Turn err.
angle	left	right	left	right	radius [m]	[deg/m]
0°	30.5	30.5	15.00	15.000	∞	0
5°	27.3	33.7	15.032	14.968	105	0.54
10°	24.0	37.0	15.065	14.935	52	1.1

road is typical, sometimes – typically close to junctions – the roll angle may reach 10°.

The effect of tyre depression may be reduced by increasing the tyre pressure, but effects from loose gravel (or soft soil), slippage and stepping effect on the tyre treads (as there is a turn momentum from the weight on the caster front wheel) influence the effective wheel radius too. All in all the heading estimate based on difference in travelled distance is systematically dependent on parameters that cannot be deduced from the odometry.

A heading error of more than $1°m^{-1}$ is not satisfactory. On a flat surface the heading stability is about $0.2°\ m^{-1}$ (based on calibration figures in Nielsen & Breiting (2004)).

Detection of roll angle could be obtained by using a tilt sensor and thereby reducing one of the error sources. A gyro measuring the heading change rate would however be able to measure the heading change independent of these error sources (but introduce others like drift as a function of temperature and time).

The used gyro reaches a heading stability matching the odometry on a flat surface at a speed of about $1\,ms^{-1}$. When the robot is stationary or at very low speed the time drifting error from the gyro exceeds the odometry error, and thus a combination of the two sensors should produce a superior result. The used implementation is a combination that uses the odometry heading when the wheels are stationary and uses the gyro when the wheels are moving. At the robot minimum speed (about $0.2\,ms^{-1}$) the gyro accuracy should still be superior to the odometry-based heading at a 10° roll angle, so a more balanced use of the odometry and gyro headings is expected to produce minor accuracy improvements only.

2.4.3 GPS

A GlobalSat GPS BU303 is available on the robot, this device supports EGNOS[1] and WAAS[2] (America only), this enables the device to obtain an about 5 m accuracy in open areas, compared to about 30 m when EGNOS/WAAS is not available (ref. Nielsen & Breiting (2004)).

2.4.4 Laser scanner

The laser scanner used is a SICK LMS-200; it is interfaced using a high speed USB-serial connection. This allows a scan rate of up to 72 scans/s with 181 measurements each scan. The used range is 8 m, at this range the absolute range accuracy is about 1 cm. The laser scanner is mounted on bas that allows

[1]EGNOS: European Geostationary Navigation Overlay Service
[2]WAAS: Wide Area Augmentation System

variable pitch. For the experiments in this thesis the tilt control is fixated at 9°. The laser scanner is mounted 41 cm above the terrain.

2.4.5 Camera

A single camera is used for vision purposes. A Philips 840K USB webcam is used, the camera is fitted with a non standard CAML12 fixed wide-angle lens. In this configuration the focal length is found to be 576 pixels for a 640×480 image.

The radial distortion constants are estimated to be $K1 = 1.29 \cdot 10^{-6}$ and $K2 = 1.78 \cdot 10^{-13}$ (K1 is a proportional factor for r^3 and K2 for r^5, where r is the radial distance (in pixels) from the image centre).

The camera is mounted at a height of 86 cm and is tilted downwards with an angle of 22°. The opening angles φ_v vertically and φ_h horizontally are calculated in (2.5)

$$\begin{aligned} \varphi_v &= 2 \tan^{-1} \frac{240}{576} = 45^\circ \\ \varphi_h &= 2 \tan^{-1} \frac{320}{576} = 58^\circ. \end{aligned} \tag{2.5}$$

The camera tilt therefore ensures – in a flat terrain – that the horizon is at the top edge of the image. This is to reduce the effects of image saturation when too much of a bright sky is visible in the image.

The out-of-band filtering – infrared and ultraviolet – was implemented as a coating on the original lens. An infrared filter is attached as a replacement to maintain a reasonable colour balance, but especially in sunshine the colour saturation is limited suggesting that the out-of-band filtering is less than optimal.

The automatic white balance filtering provided by the camera is used to adapt to the changing colour temperature under changing weather conditions. This is sufficient for the currently implemented usage, and is visually appealing. Manual operation modes are available.

2.4.6 Computer

The robot has an on-board computer (a mini ATX with a 1.2 GHz VIA processor). This integrates all sensors and the drive control systems. The operation system is Linux (in a slackware distribution) on a 1 GB flash disk. A RAM-disk is used for data logging during missions.

2.4.7 Security

The weight and speed of the platform allows it to inflict some limited damage on material and persons directly. The robot is therefore equipped with both an emergency stop switch – that removes power from the driving wheels – and a hand-held remote control capable of taking control of the vehicle whenever needed. The remote control must further be within radio coverage at all times to allow the on-board computer to control the robot.

2.5 Approach

The first attempt was to analyze the laser scanner data to separate traversable from nontraversable terrain and then follow a sequence of GPS positions to get to the destination driving on the traversable terrain only. This attempt was performed as part of the Master's thesis Riisgaard & Blas (2005), and some results were obtained. The terrain analysis used was not efficient on all the terrain types needed for this thesis, and the control strategy, in combination with the odometry-based posture sensing, was insufficient to successfully complete the navigation objectives. However the basic approach is continued in this thesis.

The objective is to navigate the different road types, as they are found in the test area, and be able to do so most of the year.

2.5.1 Road navigation

The approach is to be able to distinguish between road surface and road edges based on an analysis of the laser data supplemented by analysis of images from the camera. With the road detection in place a method should be found to drive the robot steadily on the road and to detect transitions in road topology – eg at a road junction or when entering open areas – where the control strategy should be changed.

The robot is capable of passing terrain with steps or obstacles with a height of less than 5 cm and a terrain inclination of less than 10%, this will allow it to drive on almost all roads and pass most of the obstacles found on the road, eg stones, erosions from rainfall, leaves and small branches. Some of the road edges are relative flat from wear or with cut grass; these areas are in many cases traversable by the robot, but the robot should, whenever possible, prefer the road over the grass edges.

Obstacle larger than 5 cm should be detected and avoided. Moving obstacles like pedestrians, bicycles and cars should be avoided too, but detection of moving obstacles are outside the scope of this thesis.

Some of the roads are rather wide (5 m) and here the robot should stick to normal traffic rules and keep to either the left or right side of the road, to allow passage of other traffic. On narrower roads it may be more appropriate to drive in the middle of the road.

The navigation approach is therefore to use the developed algorithms to follow a road edge or a road centre. This topological approach is independent of GPS coverage and GPS accuracy. The approach requires detection of change in topology, eg when the road ends or joins with another road. The navigation approach is thus to follow a sequence of topological features (eg roads) supplemented by odometry-controlled transitions from one road to the next when needed.

2.5.2 Vision support

This experimental platform uses a laser scanner to detect the road at a distance of about 2.5–3 m in front of the robot. This is sufficient for obstacle avoidance and road detection in most cases, but detection of junctions and obstacles at longer distances would improve navigation quality as manoeuvring could be initiated at an earlier stage. The approach is to isolate the extend of the road from images taken by a forward looking camera. The assumption is that a homogeneous road surface can be isolated from non-road areas, as the non-road areas are assumed to have a different colour or structure, or is detectable at the transition from road to non-road.

The algorithm uses a sample area in the image already classified by the laser scanner. This sample area is analyzed for colour and intensity profile, and the area in the image that matches this profile, within an allowed deviation, and not crossing clear edges, is taken as the road extend.

The reliability of this vision method is – for a number of reasons – less than the laser scanner method and is therefore used as a support function only.

2.5.3 Sensor fusion

The laser scanner classifies the terrain up to 2.5–3 m with a high probability of correct classification, the vision system has longer range, but both the accuracy and the detection probability are inferior to the laser scanner, the vision data is therefore used at ranges beyond the laser scanner range only. The basis for route planning is a traversable corridor; this corridor is based on the laser scanner traversable terrain classification and is extended using the vision data. The advantage of the extended range is that the manoeuvre planning may include avoidance of obstacles that is yet unseen by the laser scanner. In junctions the available exit roads may be visible by the vision sensor and the route therefore be adjusted in time to reach the desired exit. At places where the roadsides

are as smooth as the road itself (eg warn down grass from next to a gravelled road), the vision may be able to improve the separation.

When the vision part fails, eg is unable to find the extend of the road successfully, either finding too much (including the roadsides) or too little (hindered by eg surface change or hard shadows), the planning reduces to the laser scanner only.

The update rate of the laser scanner (about 6 Hz) is higher than the vision analysis (about 1 Hz), as the vision covers a longer range segment and therefore stays valid for a longer period of time.

2.5.4 Behaviour decisions

A number of traversable corridors may be identified, eg left and right of obstacles, side roads and main roads, or just flat areas in the roadside. A manoeuvre route is planned for all identified corridors, and the decision on which to follow is made based on the route attributes and navigation objectives.

When manoeuvring is required an attempt is made to limit the resulting centripetal acceleration maintaining the desired velocity (where possible).

When no corridor is available the robot will stop, following part of the last free route planned. If the obstacle disappears the robot will resume the route after an obstacle timeout period.

2.5.5 Software architecture

The software architecture implemented should be reusable and expandable for other than the original developers. It should not be needed to know the details of the reused code to use it, and the maintenance of the reusable code and new development should be separable. At the same time the interface points should be simple, fast and allow easy debug and function monitoring.

The component structure in Orca Brooks et al. (2005) was investigated but was not adopted, primarily for three reasons: the communication structure seemed too complicated (too much overhead for high data rate connections and too limited in data content resulting in too many connections), the extensive library dependency gave implementation difficulties in our Linux distribution, and the available components were not sufficiently attractive to overcome the other limitations.

The proposed component structure is based on a number of servers producing, where each server may have client connections to other servers. This structure, as well as the basic communication methods – server-push and client-pull – is taken from the Orca terminology. The communication media is selected to be socket-based TCP/IP. Our existing software is focused on socket-based IP-communication. The communication data formatting is selected to follow a

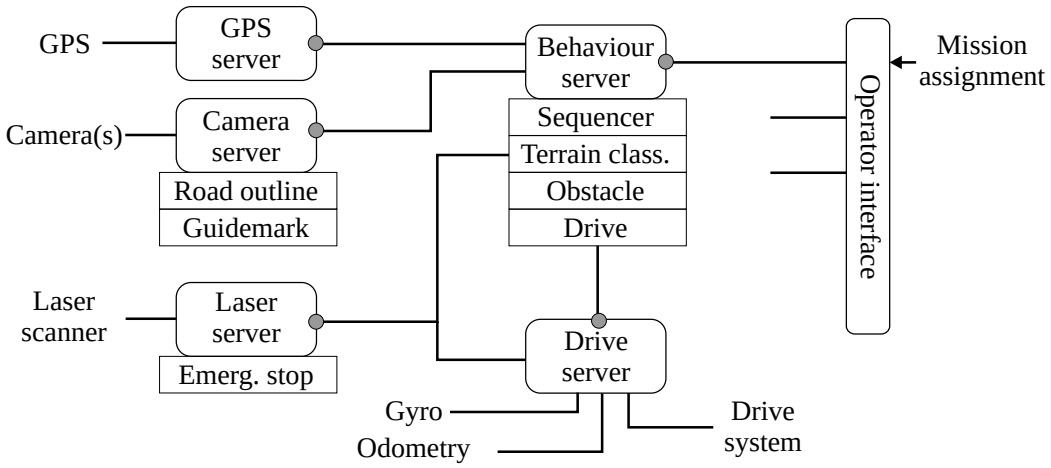


Fig. 2.5: The software architecture is based on a number of server functions that allows connection from clients (at points marked with a circle). Some of the servers allow increased functionality by plugins shown as a square attached to the server. The communication media uses IP, and the message data is formatted using XML (with a few exceptions).

subset of the XML defined in the World Wide Web Consordium (2004) recommendation. XML is selected as it is well defined, it remains compatible even if extended, it is text based allowing for interface debugging by use of simple tools, and a number of parsers and coders exist in different programming languages.

Transfer of large data structures (eg a camera image) over a socket connection is relative slow, XML coded or not, compared to data transfer internally in one application or using shared memory. An internal data transfer rate may be up to 4 bytes for each CPU clock cycle. Transfer of the same data on a socket connection is often at least 10–40 times slower.

A server handling large data structures like a camera server or a laser scanner server should therefore include as much data reduction as possible before delivering data to a client. That is, the server should hold the data analysis functionality so that only the result is transferred to the client. Much of the new data analysis functionality should therefore preferably be built into the server directly. To do this without sacrificing separate maintenance of reused code and new functionality, the new functionality should be added to the server as a plugin.

This set of selected communication and structure methods has led to the block structure shown in Fig. 2.5. Each server serves a unique purpose and may provide access to a limited set of resources – ie sensors or actuators. A server typically provides all its services on one line (shown as a small circle). Each client attached decides on the communication method and content from the

available options provided by the server. The operator interface may connect to servers for monitoring or configuration. The operator interface may be simple tools as the text-based TELNET application or more advanced with graphical data monitoring and control, possibly with capabilities like teleoperation.

Chapter 3

Laser scanner based perception

3.1 Introduction

The laser scanner is by far the most common sensor used in mobile robotics to support obstacle avoidance and path finding. The laser scanner measures direction and distance directly, and is thus an easy starting point for the navigation software. Almost all other sensors need significant processing before comparable data quality can be obtained.

Two main objectives can be fulfilled by the laser scanner in support of navigation: detection of obstacles that need to be avoided and detection of traversable areas. Detecting obstacles and defining lack of obstacles as traversable areas is one solution. The opposite – detection of traversable areas and defining non traversable areas as obstacles – is another. In an indoor environment the first method – obstacle detection only – is usually sufficient. In an outdoor environment the transition from traversable and obstacles is more fluent, and traversable areas may be more or less desirable for route planning.

Obstacles may be just obstacles – something to avoid – and traversable areas just for route planning, but they may also be used as guidemarks that can assist the navigation decisions. This will require that the obstacles or traversable areas are recognisable, in shape or position, either as unique guidemarks – like a unique signpost on a highway – or unique within a limited search area – like a house number when the search area is limited to one road.

This chapter focuses on positive sensing of traversable areas and obstacles from one and the same laser scanner sensor. The use of the data in navigation terms is deferred to a later chapter.

3.2 Related work

Current work in the area tends to focus on using 3D laser scanners or a combination of 3D laser scanners and vision.

Using 3D laser scanner solutions has been proposed by Vandapel et al. (2004) by transforming point clouds into linear features, surfaces, and scatter. These were classified by using a Bayesian filter based on a manually classified training set.

Identification of navigable terrain using a 3D laser scanner by checking if all height measurements in the vicinity of a range reading had less than a few centimetres deviation is described in Montemerlo & Thrun (2004).

An algorithm that distinguished compressible grass (which is traversable) from obstacles such as rocks using spatial coherence techniques with an omnidirectional single line laser is described in Macedo et al. (2000).

A method for detection and tracking the vertical edges of the curbstones bordering the road, using a 2D laser scanner, described in Wijesoma et al. (2004) is a way of indirect road detection.

Detection of borders or obstacles using laser scanners is often used both indoors and in populated outdoor environments, and is the favoured method when the purpose includes map building, as in Guivant et al. (2001) and Klöör et al. (1993).

Detection of nontraversable terrain shapes like steps using laser scanner for planetary exploration is described in Henriksen & Krotkov (1997).

The DARPA Grand Challenge 2004 race demonstrated the difficulties in employing road following and obstacle avoidance for autonomous vehicles Urmson et al. (2004). This situation seems to be improved in the 2005 version of the race, where five autonomous vehicles completed the 212 km planned route. The winning team from Stanford perceived the environment through four laser range finders, a radar system, and a monocular vision system. Other teams, like the gray team Trepagnier et al. (2005) also use laser scanners as the main sensor for traversability sensing supplemented by (stereo) vision. The solution of the winning team in 2005 is described in Thrun et al. (2006); a 2D laser scanner detects traversable road based on the vertical distance between measurements, this solution is combined with vision and radar for longer range detections.

3.3 Overview

As in most of the related work, the primary navigation sensor for this thesis is the laser scanner. It supports the two main drive methods: *road following* and a *direct to destination* mode.

- Road following attempts to keep a fixed position relative to the road, or aims for a fixed position following the most appropriate of the detected roads.

- Direct to destination mode is intended for tight manoeuvres where the road information is either irrelevant or just seen as obstacles.

The same laser scanner data is used for both road finding and for obstacle detection.

- The road finding is based on a series of connected traversable segments from successive laser scans; this series is assumed to form a corridor that describes the road. The traversable segments are found by analysing the laser scanner data for a set of features that combined describes traversable terrain.

- Obstacles are found in the part of the laser scanner measurements classified as nontraversable. These single scan obstacles are then combined with obstacles found in previous scans into confirmed obstacles. Two obstacle modes are used when obstacles are correlated with confirmed obstacles: indoor or outdoor.

The laser scanner positioning and scan rate requirements are discussed first, and then the data is analyzed for traversable corridors and for obstacles.

The use of the detected road corridor and obstacles are deferred to chapter 5.

3.4 Laser scanner use

To detect the road the laser scanner must be tilted towards the road. This further allows detection of obstacles up to the road detection distance.

The optimal positioning and tilt of the sensor is a compromise between laser scanner sensitivity and range, the required obstacle warning distance, the minimum obstacle size and the required scan rate. The used setup is shown in Fig. 3.1.

The longer warning distance required the smaller the angle between the laser beam and the road surface. A small angle reduces the amount of returned laser light and thus reduces sensitivity. This is especially a problem on a wet surface, eg pits filled with water and wet asphalt. A small angle and a high scan rate both results in better obstacle detection capability and longer obstacle detection range. A higher laser scanner mounting height results in a longer warning distance. The laser scanner range – 8 m in the used mode – further sets a limit for the angle of tilt.

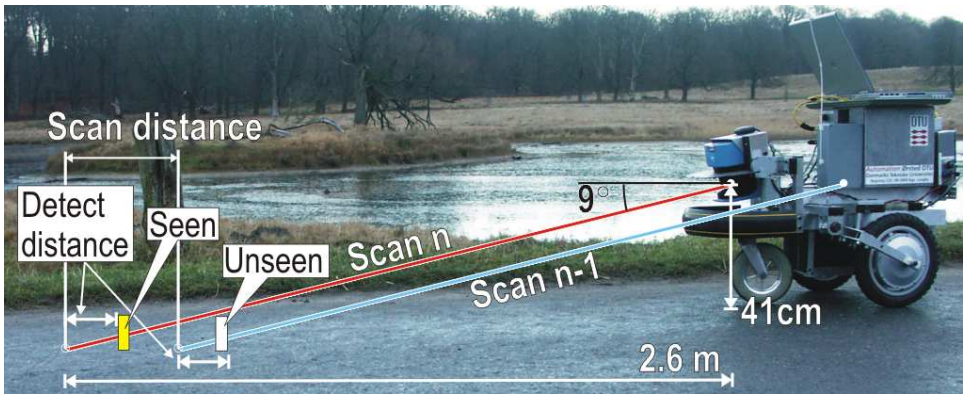


Fig. 3.1: The robot used in the experiments has a laser scanner at a height of 41 cm looking downwards in an angle of $\theta_L = 9°$. A flat road is therefore detected at 2.6 m. The position of the last scan (scan $n-1$) is shown to illustrate the largest undetected obstacle. The unseen object is not detected in scan $n-1$ and is too short to be seen by scan n.

3.4.1 Obstacle detection

The largest obstacle D_o that can remain undetected is a function of robot velocity V, the tilt-angle of the laser scanner Θ_L, the scan rate f_s and the distance x_j required to separate the obstacle from the road. The obstacle size D_o can be calculates as shown in (3.1), or as a limitation on the scan rate in (3.2)

$$D_o = (\frac{V}{f_s} + x_j) \tan \Theta_L \tag{3.1}$$

$$f_s = V/(\frac{D_o}{\tan \Theta_L} - x_j). \tag{3.2}$$

This scan rate f_s corresponds to a required distance between scans D_s maintaining the obstacle detection capability

$$D_s = \frac{V}{f_s} = \frac{D_o}{\tan \Theta_L} - x_j. \tag{3.3}$$

For an obstacle size of $D_o = 5$ cm, a detection distance of $x_j = 15$ cm and a laser scanner tilt-angle of 9° the required distance between scans is 16 cm.

Further the obstacle detection distance $x_j = 15$ cm has the consequence that obstacles down to 2.5 cm ($x_j \tan(9°)$) may be detected as obstacles. This means that obstacles that can be ignored may trigger an unnecessary manoeuvre.

3.4.2 Security distance

The maximum allowable robot velocity $V_{\max}$ depends on the maximum brake acceleration a_b and the available brake distance S_b as shown in (3.4)

$$V_{\max} = \sqrt{2a_b S_b}. \tag{3.4}$$

The available breaking distance S_b is shorter than the road detection distance S_r by: a guard distance S_g, the distance required to detect the obstacle x_j, the distance travelled in the reaction time t_r and the scan rate f_s, as shown in (3.5)

$$\begin{aligned} S_b &= S_r - S_g - x_j - V(t_r + f_s^{-1}) \\ S_r &= \frac{h_L}{\tan \Theta_L}, \end{aligned} \tag{3.5}$$

where h_L is the laser scanner height.

3.4.3 Scan rate requirement

(3.5) and (3.4) can be expressed as a scan rate requirement as shown in (3.6)

$$f_s = V(S_r - S_g - x_j - Vt_r - \frac{V^2}{2a_b})^{-1}. \tag{3.6}$$

The scan rate requirements are shown in Fig. 3.2 for three different laser scanner tilt-angles: 6° (a), 9° (b) and 11° (c), all with a laser scanner height of 41 cm. A breaking acceleration of $2\,\mathrm{ms}^{-2}$ is about the maximum obtainable on a gravelled surface.

The maximum usable cruise speed for the robot is about 1.5–$2\,\mathrm{ms}^{-1}$ (5.4–$7.2\,\mathrm{kmh}^{-1}$). For $1.5\,\mathrm{ms}^{-1}$ the detection range and required scan rate are shown in table 3.1 for the tree cases shown in Fig. 3.2.

Table 3.1: The scan rate and detection range are a function of the tilt-angle of the laser scanner and the robot velocity.

	$1.5\,\mathrm{ms}^{-1}$			$2\,\mathrm{ms}^{-1}$			
Scanner tilt	6°	9°	11°	6°	9°	11°	unit
Obstacle detect	5	9	14	6	12	18	scan/s
Emergency stop	1	1.5	2.5	1	4	57	scan/s
Detect range	3.9	2.6	2.1	3.9	2.6	2.1	m

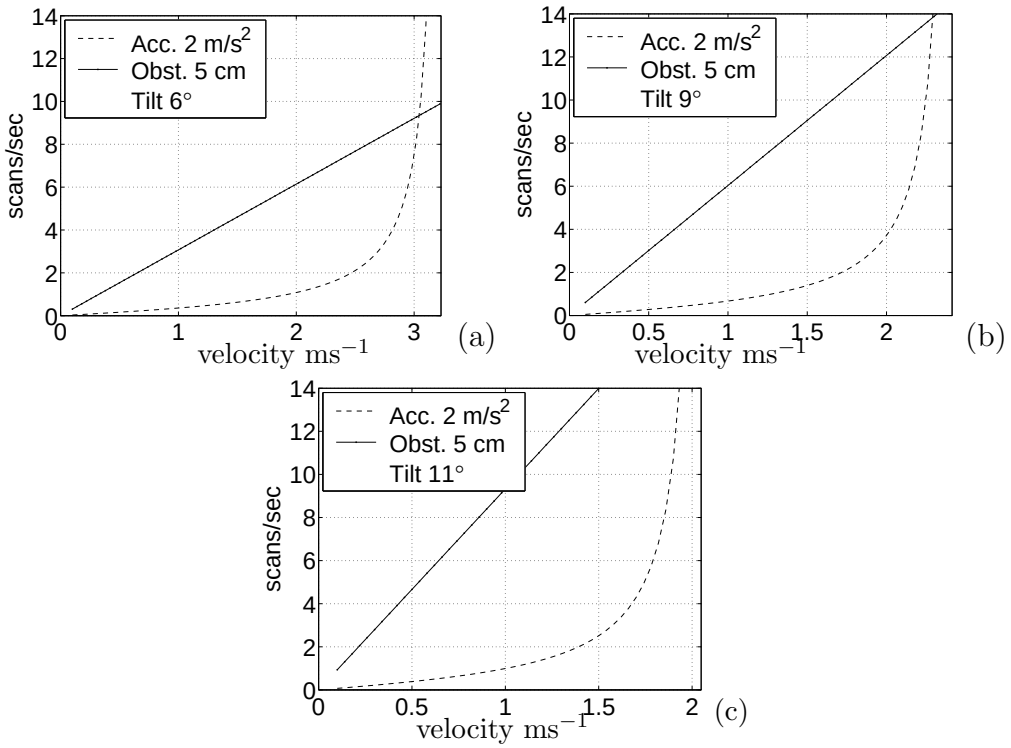


Fig. 3.2: The required scan rate is a function of robot speed, tilt angle of scanner and detected obstacle size. At a tilt-angle of 9° (b) and a speed of $1.5\,\mathrm{ms}^{-1}$ a scan rate of 9 Hz is required for obstacle detection, whereas 2 Hz is sufficient for emergency stop behaviour.

3.4.4 Laser scanner tilt

A tilt of 11° provides a good reflection angle for the laser scanner, but will require a processing scan rate of 57 scans/s at a velocity of $2\,\mathrm{ms}^{-1}$. The laser scanner can provide up to 72 scans/s, but this high scan rate will be significant for the on-board processor (classification and obstacle detection take about 4 ms, and with 57 scans/s this is about 23% of the processing time available).

A tilt of 6° has a low reflection angle for the laser scanner and will not detect the road if the road curves have an inclination change of more than 6° over the maximum detection range (8 m). Such change of terrain curvature may be found on parts of the target route.

A tilt of 9° may be the right compromise between scan rate requirement, road detection in wet weather and the terrain curvature found in the target terrain. Therefore this angle of tilt is analysed further.

In the curved terrain shown in Fig. 3.3 the inclination changes from flat to −9° resulting in a road detection distance of 7 m. Seen from a robot perspective the detected surface is 70 cm lower than the robot base. A corresponding

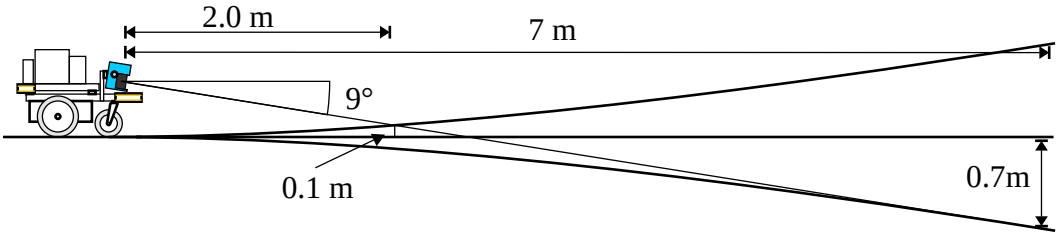
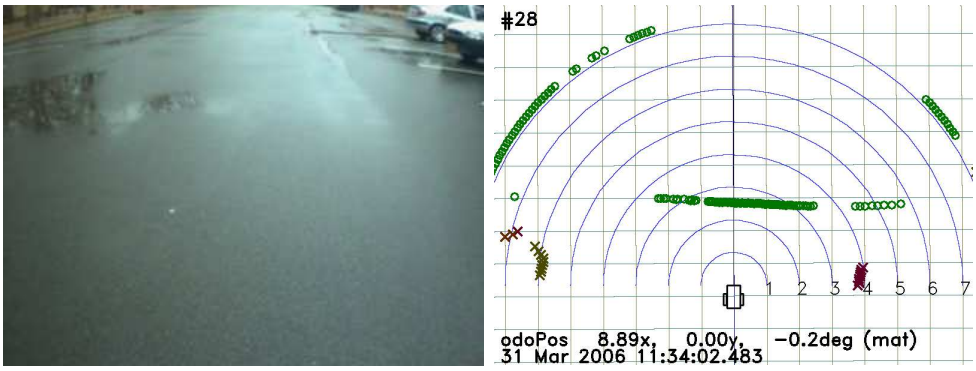


Fig. 3.3: The road inclination may change from flat and down to about −9° in just 7 m, and the road will still be detected. A corresponding increase in inclination reduces the detection range to about 2 m.



(a) (b)

Fig. 3.4: The wet parking lot (a) seen by the robot camera is detected by the laser scanner as shown in (b). The small circles and crosses in (b) are the laser measurements. In some areas left and right the returned laser beam is too week to be detected. The nonvalid measurements are returned as maximum range (just above 8 m). The area to the left – where the detections start to fade – corresponds to the more wet area in image (a). The measurements (crosses) left and right are parked cars not visible in the camera image (a).

increase in inclination results in a road detection distance of about 2.0 m (compared to 2.6 m for a flat road).

An obstacle with a height of 5 cm below the front or rear wheels will result in a robot tilt of about ±6.5° of the robot. At a nominal laser tilt of 9° the resulting tilt of the laser scanner will be from 2.5° to 15.5°. An otherwise flat surface will in this case be detected at a distance from 10 m (beyond laser scanner range) down to 1.5 m. Detection of a road at just 1.5 m corresponds to a terrain elevation of 17 cm when the laser scanner tilt is assumed to be 9°.

3.4.5 Wet surface reflection

On a wet surface – especially pits filled with water and wet asphalt – the used laser scanner has difficulties in detecting the surface at the narrow angles used. An example is shown in Fig. 3.4(a) from a wet parking lot where there are signs of blank water in the left side of the image; the laser scanner measurements fails in parts of this area, as shown in Fig. 3.4(b). Just in front of the robot and at some areas to the right, the wet surface is detected.

3.4.6 Spurious detections

The laser scanner is seen to return spurious detections, the origin of the spurious detections may be raindrops, insects, snowflakes or just spurious. Typically there is only one detection at the particular position and it is not repeated in the following scan.

A thin fence pole or a chair leg are also often seen by one measurement only, so to avoid the spurious detections isolated measurements are to be trusted as obstacles if they persist in more than one scan only.

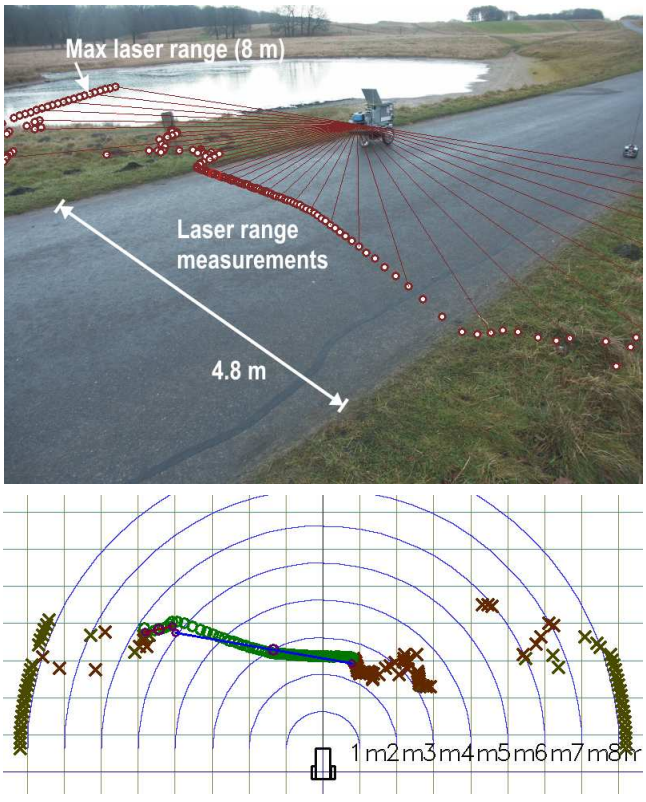
3.4.7 Summary

A laser scanner tilt of 9° is about right, it allows detection of terrain curvature to the degree found in the target area, it allows a manageable scan rate of about 9 scans/s for obstacle detection at a velocity of 1.5 m/s, and 2 scans/s for manoeuvre planning (especially emergency stopping). When the robot climbs obstacles, the robot tilt will add to the laser scanner tilt temporarily, and obstacles larger than the allowed size may pass unnoticed at the calculated scan rate. The calculated scan rate for obstacle detection should therefore be exceeded to allow such rough conditions. As the tilt variations due to obstacles are assumed to be temporary, the effect on the required scan rate for manoeuvre planning is not expected to be influenced significantly.

The wet asphalt problem is not fully solvable, but as shown in Fig. 3.4 the road just in front of the robot is usually detected and, if the missing range measurements are ignored, a reasonable road width can be detected.

3.5 Traversability

The assumption is that the measurements can be divided into traversable and nontraversable based on the measurement range values from one single laser scan. By looking at the measurements in Fig. 3.5 this assumption seems reasonable. A number of exceptions need to be considered though, like rough gravelled road surfaces, relative smooth road sides, stones and branches on



(a)

(b)

Fig. 3.5: Measurements from the laser scanner from an asphalt road. The measurements are – as expected – more homogeneous from the road surface than from the grass edges. On the robot top view (b) the circles are measurements classified as road (traversable), and the crosses as nontraversable. From the measurements it is clearly seen that the road is about 5 m wide and slightly curved.

the road, ditches and the road profile, as well as wet reflecting surfaces and pits filled with water. The traversability classification is therefore a bit more complicated than might be expected.

The traversability analysis described here is accepted for publication in the journal paper Andersen et al. (2006). The work was initiated in the master's thesis Riisgaard & Blas (2005), improved in the paper Blas et al. (2005) and now further improved as described here.

3.5.1 Measurements

With laser scan readings given in the plane of the laser sensor, each laser scan $P^L(d_i, \phi_i)$ comprises a set of range readings d_i, $i \in [1, N]$ and the angles $\phi_i \in [-90°, 90°]$ associated with these. A laser scan P^L is hence represented in polar coordinates in the laser scanner frame. A single element of P^L is denoted P_i^L. A measurement can be transformed to a robot fixed frame by an $\mathbb{R}^2 \rightarrow \mathbb{R}^3$ mapping $\mathcal{M}_{RL}(h_L, \theta_L) : P^L(d_i, \phi_i) \rightarrow P(x_i, y_i, z_i)$ as shown in (3.7). The robot frame has body-fixed axes x, y, z pointing forward, left and up, respectively. Scanner measurement angle ϕ_i is zero forward and positive to the left

$$P_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} d_i \cos\Theta_L \cos\phi_i \\ d_i \sin\phi_i \\ h_L - d_i \sin\Theta_L \cos\phi_i \end{bmatrix}. \tag{3.7}$$

With a constant laser tilt-angle Θ_L the coordinates x_i and z_i are fully correlated as $z_i = h_L - x_i \tan\Theta_L$, and thus carry the same terrain information. In the calculations below the x_i values are used primarily, as this value is directly visible in an x–y map projection, but especially if a controlled tilt was to be implemented, the z_i values would be preferable over x_i as these are more independent of the laser scanner tilt.

In the sequel, $P^L(k)$ will denote a full set of range measurements from a single laser scan at time k, and $P(k)$ the scan converted to the robot fixed, cartesian frame.

3.5.2 Feature membership functions

The description is based on two main assumptions, a classification assumption and a mapping assumption:

Classification assumption: Each 3D reading can be categorized as belonging to either of three classes, $P_i(k) \in \mathcal{C}$, $\mathcal{C} = \{\mathcal{C}_t, \mathcal{C}_n, \mathcal{C}_\emptyset\}$ where $\mathcal{C}_t$: traversable, $\mathcal{C}_n$: nontraversable and $\mathcal{C}_\emptyset$: invalid data.

Each $P_i(k) \in \mathcal{C}_t$ will belong to one traversable segment $\mathcal{S}_j(k)$ of the set of traversable segments $\mathcal{S}$ in the scan k.

Mapping assumption: Seven main features characterise the natural environment, : $\mathcal{F}_h$: raw height, $\mathcal{F}_\sigma$: roughness, $\mathcal{F}_z$: step size, $\mathcal{F}_c$: curvature, $\mathcal{F}_w$: slope and width, and $\mathcal{F}_\emptyset$: invalid data. A unique mapping exists $\mathcal{M}_{CF}(\mathcal{F}) : \mathcal{F} \rightarrow \mathcal{C}$ that categorisation can be uniquely determined from.

Further a mapping exists $\mathcal{M}_{SF}(\mathcal{F}) : \mathcal{F} \rightarrow \mathcal{S}$ for measurements in category $\mathcal{C}_t$.

Testing each measurement P_i and associating a membership function with each feature, the aim is to determine a classification into traversable and non-traversable segments and let this information navigate the robot. The formal procedures of feature extraction and test of hypotheses about categorisation are pursued in the following subsections, starting with a formal statement of the problem. It is a prerequisite that scan readings $P^L(k)$ are transformed to the appropriate body coordinates $P(k)$. Let $\mathcal{F}_i \subseteq \{\mathcal{F}_h, \mathcal{F}_\sigma, \mathcal{F}_z, \mathcal{F}_c, \mathcal{F}_w, \mathcal{F}_\emptyset\}$ and $\mathcal{C}_i \in \{\mathcal{C}_t, \mathcal{C}_n, \mathcal{C}_\emptyset\}$,

problem: Given 2D laser readings $P(k)$, determine membership of a feature function $\mathcal{F}_i(P_i(k))$ and determine a mapping $\mathcal{M}_{CF}(\mathcal{F}) : \mathcal{F} \rightarrow \mathcal{C}$ to categorise $P_i(k) \in \mathcal{C}_i$, and for $P_i \in \mathcal{C}_t$ a mapping $\mathcal{M}_{SF}(\mathcal{F}) : \mathcal{F} \rightarrow \mathcal{S}$.

For the sake of brevity, and because feature extraction and classification are done on single scans in this context, the scan index k is omitted in the remainder of this section.

3.5.3 Invalid data

The laser scanner detects the surface at a small angle, but in most cases the surface is sufficiently rough to get stable range measurements. On smooth surfaces the laser scanner is often unable to detect the surface, as shown in section 3.4.5.

In these cases the laser scanner returns maximum range, the same value as if the reflecting surface was further away than the maximum range (8 m).

The feature extraction for invalid data is

$$\mathcal{F}_\emptyset(P_i^L) = \{d_{\text{inf}} \leq d_i\}\,. \tag{3.8}$$

The limit is set to the maximum range for the laser scanner $d_{\text{inf}} = 8\,\text{m}$.

These measurements are now classified and are disregarded from the remaining classification process.

3.5.4 Raw height feature

The feature extraction for valid data starts with the raw height of the measurement, $h(P_i)$

$$\mathcal{F}_h(P_i) = \{\ h_{\text{inf}} < h(P_i) < h_{\text{sup}}\}\,. \tag{3.9}$$

The limits h_{inf} and h_{sup} depend on the expected terrain variations as described in section 3.4.4 and further on the robot sensitivity to tilt and yaw.

This $h_{\text{sup}} = 0.2\,\text{m}$ threshold will limit erroneous classification of flat surfaces, eg a wall in front of the robot, and allow true road detection in a reasonable combination of robot tilt and road curvature. The lower threshold is more sensitive to road curvature in combination with tilt and yaw of the robot. The lower limit was set to $h_{\text{inf}} = -0.7\,\text{m}$ in the experiments; this should ensure road detection in the road curvature situations of the test area – see Fig. 3.3 – but not in all combinations of robot tilt and road curvature. The robot tilt situation is expected to be of temporary nature, so a situation without road detection should be temporary and thus recoverable when the robot tilt is back to normal.

3.5.5 Roughness feature

Roughness of data in a 2D laser scan is defined as the square root of local variance of distance to reflections along the scan. A general estimation of local roughness was given in Blas et al. (2005) using a singular value decomposition approach. However, an alternative and less computationally demanding algorithm were implemented as described below.

Roughness algorithm

Select a consecutive point set $P_{n,m}$ from P_n covering a width $W_{\text{fit}} = B$ or wider to the nearest measurement. B is the wheel base (0.45 m).

$$\begin{aligned} P_{n,m} = &\left\{ P_i \in P \;\middle|\; \begin{array}{l} n \le i \le m \\ m = m_{\min} \end{array} \right\} \\ &\text{where} \\ m_{\min} = &\,\text{MIN}(m) \;\middle|\; \begin{array}{l} \mid P_m - P_n \mid \ge W_{\text{fit}} \\ \wedge\ m - n \ge 3 \end{array} \end{aligned} \tag{3.10}$$

Calculate the best fitted line L_n in the x, y plane for the points in $P_{n,m}$ as shown in (3.11). A least square fit of the deviation in the x-direction is used

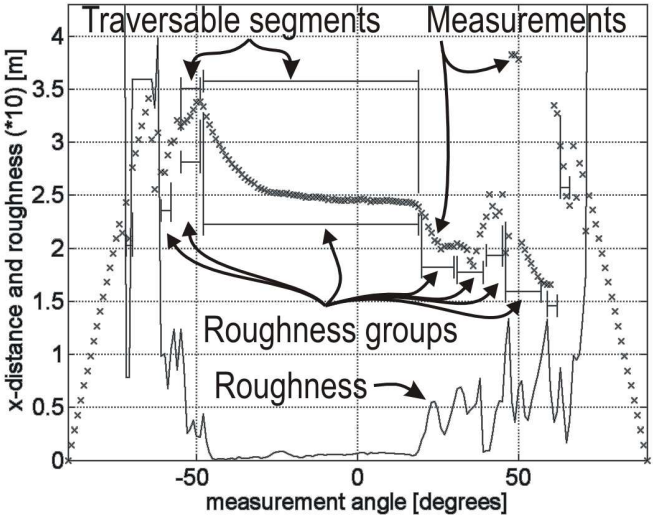


Fig. 3.6: The laser measurements from Fig. 3.5 are shown in measurement order ($-90°$ to $90°$) in order to show the roughness estimate at each angle.

for simplicity.

$$L_n = \left\{ x = a_n + b_n y \;\middle|\; \begin{array}{l} a_n = \overline{x} - b\overline{y} \\ b_n = \sigma_{xy}^2 \sigma_{yy}^{-2} \\ \sigma_{xy}^2 = \sum (x_i - \overline{x})(y_i - \overline{y}) \\ \sigma_{yy}^2 = \sum (y_i - \overline{y})^2 \end{array} \right\} \tag{3.11}$$

Calculate roughness R_n in the $P_{n,m}$ interval as the average deviation from the fitted line

$$R_n = \sqrt{\frac{1}{m-n} \sum_{i=n}^{m} \frac{(b_n y_i - x_i + a_n)^2}{b_n^2 + 1}}. \tag{3.12}$$

The length of the fitted line $W_{\text{fit}} = B = 0.45\,\text{m}$ is selected to emphasise terrain variation with a spatial period shorter than this distance and wide enough to allow at least three measurements – in most cases – within this distance. Soft curves with a spatial period longer than the robot width will result in a relatively small roughness value. An example of the roughness value is shown in Fig. 3.6 for the scene from Fig. 3.5. The roughness is the full line at the bottom, and the corresponding laser scanner measurements x_i are shown above.

Roughness groups

The intention is to group measurement into traversable segments. If there are more types of traversable segments these should be grouped separately. An example of traversable segment types could be an asphalt road edged by cut grass, both segment types are traversable, but it would – in most cases – be preferable for the robot to keep to the asphalt. This is accomplished in three steps. First the measurements are divided into homogeneous groups (using height and roughness). These groups are then combined using the classifiers for step size $\mathcal{F}_z$ and curvature $\mathcal{F}_c$. Finally the resulting groups are filtered based on slope and width $\mathcal{F}_w$ to a set of traversable segments $\mathcal{S}$.

Neighbouring traversable points P_i in P_n to P_m are said to belong to group G_j if their roughness values R_i are relatively homogeneous – as defined in (3.13)

$$
\begin{aligned}
G_j = & \\
& \left\{ P_i \in P_{n,m} \;\middle|\; \begin{array}{l} h_{inf} < z_i < h_{sup} \\ \wedge\, R_i < R_{sup} \\ \wedge\, E(P_{n-1}, L_n) < R_j^{\lim} \\ \wedge\, E(P_{m+1}, L_n) < R_j^{\lim} \end{array} \right\} \\
& \text{where} \\
& E(P_i, L_n) = \left| \frac{b_n y_i - x_i + a_n}{\sqrt{b_n^2 + 1}} \right| \\
& R_j^{\lim} = \alpha \, \mathrm{MIN}(R_i \,|P_i \in P_{n,m}\,).
\end{aligned}
\tag{3.13}
$$

This implement an adaptive threshold based on the point with the minimum roughness. The interval is then expanded to the point where the distance to the next measurement is above the adaptive threshold. The distance $E(P_i, L_n)$ is taken from the next measurement P_{m+1} or P_{n-1} to the fitted line L_n used to calculate R_n. The threshold limit $R_j^{\lim}$ is set to α times the minimum roughness inside the interval.

The value of the roughness threshold $R_j^{\lim}$ was found experimentally to be $\alpha = 4.5$ by optimizing for large group size and to avoid combination of different terrain types.

Finally, association of a point to the feature $\mathcal{F}_\sigma$ is obtained as

$$\forall j : \mathcal{F}_\sigma(P_i) = P_i \in G_j \tag{3.14}$$

Figure 3.6 is an example of the resulting grouping, using data from the asphalt road with rough grass edges shown in Fig. 3.5. The groupings are shown below the laser measurements. The groupings include the asphalt area (from about $-50°$ to $20°$), as well as a number of short segments in the rough

grass. The asphalt area has a very low roughness (less than 0.01 m), but aligned measurements, with a roughness below the hard threshold $R_{\text{sup}} = 0.1\,m$, are also found outside the road area.

3.5.6 Step size

The surface of gravelled roads and areas are often a combination of smooth areas separated by small objects like stones or tracks. The adaptive threshold typically separates such intervals. Branches, leaves and stones also tend to break up otherwise smooth roads into separate segments.

The produced groups are therefore inspected and flagged for possible combination if they are likely to be from the same surface and the area between the groups is traversable.

The step size feature ensures that two segments are not combined if they are separated by an obstacle, or separated in height, ie a sidewalk should not be combined with the road.

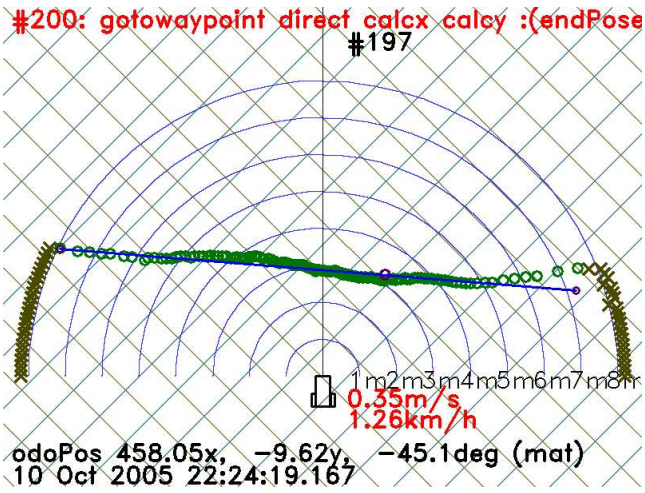
Included in the step size feature $\mathcal{F}_z$ is also a step in roughness, as defined in (3.15); this prohibits combination of a segment representing a road with a segment representing a traversable roadside (eg cut grass) into just one traversable segment.

Further, group G_j (with measurements from P_n to P_m) may be combined with group G_{j+1} (with measurements P_p to P_q) under the following condition

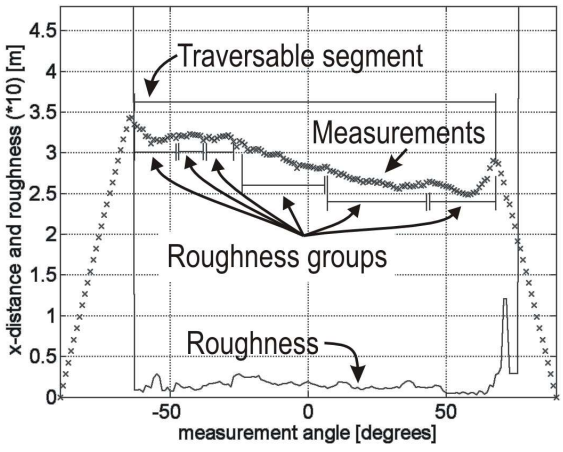
$$\mathcal{F}_z(P_i) = \left\{ \begin{array}{c|l} P_i \in G_j \\ P_i \in G_{j+1} \\ p > m \end{array} \begin{array}{l} p - m \leq 4 \\ \wedge\, R_{\text{lim}}^{-1} < \left(\frac{R_k}{R_{k+1}}\right)^2 < R_{\text{lim}} \\ \wedge\, |x_m - x_p| < D_{\text{lim}} \\ \wedge\, |x_d - \overline{x}_g| < D_{\text{lim}} \\ \text{where} \\ x_d \in P_{m+1,p-1} \\ \overline{x}_g = \frac{x_m + x_p}{2} \end{array} \right\}. \tag{3.15}$$

This step criterion allows groups to be separated by up to 3 measurements, as long as the x value of these measurement points P_{m+1} to P_{p-1} are not too far away ($D_{\text{lim}} = 15\,\text{cm}$) from the average of the two end points. The groups may not be combined if the roughness R_j and R_{j+1}, associated with groups G_j and G_{j+1}, differ by more than a factor $R_{\text{lim}} = 1.8$. The x difference between the near end points of the groups may be separated by no more than 15 cm (corresponding to a difference in height of 2.5 cm with the sensor tilt chosen).

An example is shown in Fig. 3.7. This is from a rather flat gravelled area. The area was divided into 6 roughness groups, but all were allowed to be combined.



(a)



(b)

Fig. 3.7: Measurements from a relatively flat gravelled terrain is shown relative to the robot in (a). The traversable area is divided into a number of roughness groups, as shown below the measurement points in (b). These groups are then combined to one traversable segment (shown above the measurement points).

3.5.7 Curvature

A curvature criterion ensures that a road with a high convex profile can be classified as one traversable segment, whereas concave profiles that often describe a ditch cannot.

The feature $\mathcal{F}_c$ defined in (3.16) checks for curvature and allows combination of two groups G_j ($P_i \in P_{n,m}$) and G_{j+1} ($P_i \in P_{p,q}$) based on the vertical angle between the groups. The vertical angle of the two groups may be concave with an angle of no more than $\Delta_{\text{sup}} = 3°$. For convex groups, the criterion is $\Delta_{\text{inf}} = -10°$

$$\begin{aligned}
&\mathcal{F}_c(P_i) = \\
&\left\{ \begin{array}{c} P_i \in G_j \\ P_i \in G_{j+1} \\ p > m \end{array} \;\middle|\; \begin{array}{l} A_{j+1} - A_j < \Delta_{\text{sup}} \\ \wedge\, A_j - A_{j+1} < \Delta_{\text{inf}} \end{array} \right\} \\
&\text{where} \\
&A_j = \tan^{-1}\left(\frac{z_m - z_n}{y_m - y_n}\right) \\
&A_{j+1} = \tan^{-1}\left(\frac{z_q - z_p}{y_q - y_p}\right).
\end{aligned} \tag{3.16}$$

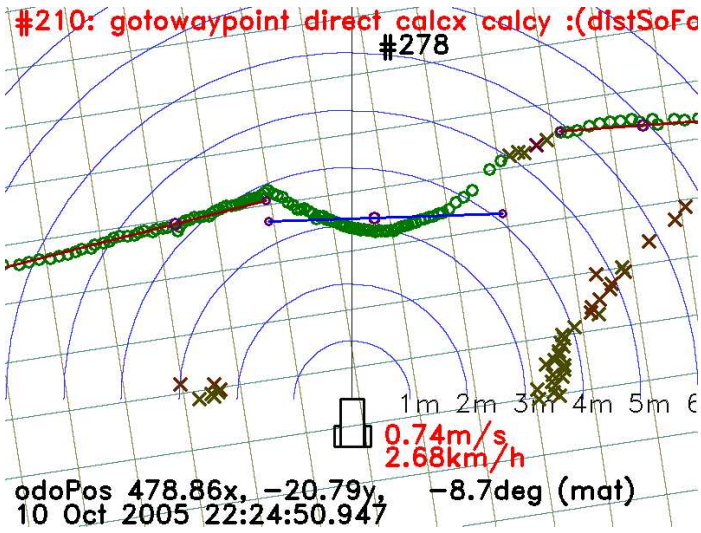
An example of this group combination and prohibited combination can be seen in Fig. 3.8, where the high profile road is crossed by a bridle path. The bridle path is separated from the road, but both parts are smooth enough to be classified traversable.

3.5.8 Slope and width

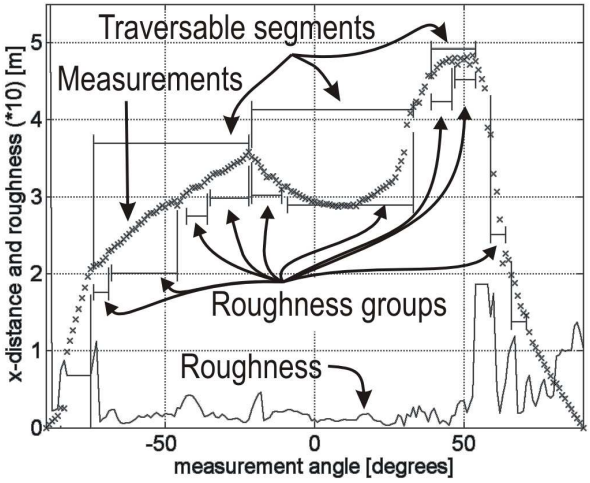
Consecutive groups formed by points that posses the features $\mathcal{F}_z$ (step) and $\mathcal{F}_c$ (curvature) are merged to one group G_j. Each group G_j (with measurements $P_i \in P_{p,m}$) must further fulfil a combined slope and width criterion to become fully qualified traversable segments $\mathcal{S}_j$.

The slope of a traversable segment must be less than $\Delta_{yz} = 10°$ vertically (in the y-z plane), and the width of a traversable segment must be wider than the robot (W_R =robot width)

$$\begin{aligned}
&\mathcal{F}_w(P_i) = \left\{ P_i \in G_j \;\middle|\; \begin{array}{l} |P_m - P_p| > W_R \\ \wedge\, \mid A_j \mid < \Delta_{yz} \end{array} \right\} \\
&\text{where} \\
&A_j = \tan^{-1}\left(\frac{z_m - z_q}{y_m - y_q}\right).
\end{aligned} \tag{3.17}$$



(a)



(b)

Fig. 3.8: Data from a gravelled road crossed by a bridle path. The road is the area with the high profile (about 15 cm higher at the centre). There are relatively flat areas from the bridle path on both sides. Rough grass on the path edges before the bridle path is just visible on both sides of the robot (a). The segmentation algorithm separates the road from the bridle path, and marks both as traversable (b). The resulting segments are shown above the measurements.

The groups that are left out by the criterion $\mathcal{F}_w$ are typically originating from a few aligned measurements in the rough vegetation. In Fig. 3.6 the groups in the rough edges are all removed with the exception of a relatively flat – and probably traversable – area at the right roadside. The resulting traversable segments are shown above the measurement points.

3.5.9 Single scan classification

The classification of points of the single laser scan is finally obtained as

$$\begin{aligned} \mathcal{C}_{\emptyset}(P_i) =& \mathcal{F}_{\emptyset}(P_i) \\ \mathcal{C}_t(P_i) =& \mathcal{F}_h(P_i) \cap \mathcal{F}_\sigma(P_i) \cap \mathcal{F}_z(P_i) \cap \\ & \mathcal{F}_c(P_i) \cap \mathcal{F}_w(P_i) \\ \mathcal{C}_n(P_i) =& \complement \left(\mathcal{C}_t(P_i) \cup \mathcal{C}_{\emptyset}(P_i) \right) \ , \end{aligned} \tag{3.18}$$

and the formation of traversable segments from the defined groups as

$$\mathcal{S}_j = \mathcal{F}_w(P_i \in G_j) \ . \tag{3.19}$$

3.6 Road detection

Road detection is used for two purposes:

- To determine where the robot should be positioned on the road, this could be on the left side, the right side, at the centre of the road, or at some distance from these road lines.
- To get road information usable for planning decisions, especially the width, eg the road width could indicate that the robot has entered a junction, or has entered a narrow road.

A quality value is associated with the road lines (left, right and centre) as well as the width; this quality value is indicating the steadiness of the lines. These quality values can be used as an indication of the fact that the road edges are valid and not just parked bicycles or other obstacles.

The number of traversable segments in the most recent scan is used as an indication of the number of possible roads or traversable corridors visible in front of the robot. More than one segment may be found, it could be: a sidewalk and a road as shown in Fig. 3.9, a road split, a road with traversable roadsides or just an error. In any case the road quality and roughness values will be valuable, when the robot has to determine where to go.

Fig. 3.9: View from the robot camera while driving on a bicycle path with the road to the left and a sidewalk to the right. The laser scanner measurements are shown in Fig. 3.10.

The road is detected by correlating traversable segments over a number of scans from the most recent and some distance back. Each set of correlated segments is called a corridor. The road lines are then calculated from the left, right and centre of these corridors.

The corridors – possibly extended by vision-based road detection (see chapter 4) – are further used when determining the best route for the robot to follow, as described in chapter 5.

3.6.1 Segment correlation

The segments found in laser scan k is $\mathcal{S}_j^k \mid j \in [1, m]$, where m is the number of traversable segments found in scan k, m may typically range from 1 to 3 with extremes being 0 (no traversable segment) and up to 4–5.

A segment S_j^k may correlate with a corresponding segment in the previous scans to form a corridor

$$B_i = \left\{ S_j^k, S_{j_1}^{k-1}, S_{j_2}^{k-2}, ..., S_{j_n}^{k-n} \right\} \tag{3.20}$$

formed by segments from scan k back to scan $k - n$.

Each segment is represented by a parameter line segment of length l

$$S_j = \{ \mathbf{x} + \mathbf{v}t \mid t \in [0, l] \}, \tag{3.21}$$

where $\mathbf{x} = [x_x, x_y]^T$ is the (right) end of the segment and $\mathbf{v} = [v_x, v_y]^T$ is a unit vector in the direction of the line fitted for the segment. The parameter t is used to represent a position $\mathbf{s}_j(t)$ on the segment for $t \in [0, l]$.

The correlation of two line segments is accepted if there is an overlap of at least the robot width W_R between the segments. The overlap is tested by projecting $t_0 \rightarrow t_1$, where t_0 is a position on segment S_j^k and t_1 the projected

position on segment $S_{j_1}^{k-1}$; this projected point or the reversely projected point $t_1 \rightarrow t_0$ should overlap with at least the robot width.

The projected position $t_1(t_0) = t_0 \rightarrow t_1$ of the point $\mathbf{s}_j(t_0)$ from segment S_j^k on segment $S_{j_1}^{k-1}$ is

$$t_1(t_0) = \mathbf{v}^T \cdot (\mathbf{s}_j(t_0) - \mathbf{x}). \tag{3.22}$$

The overlap is tested by projecting both the end point $t_a = 0$ and a robot width into segment $t_a = W_R$, both projected positions should be inside the other segment, or alternatively from $t_a = l_a$ to $t_a = l_a - W_R$ should be inside the other segment. The complete test function $\mathcal{F}_o(S_{j_a}, S_{j_b})$ for overlap between segment S_{j_a} and S_{j_b} is shown in (3.23)

$$\mathcal{F}_o(S_{j_a}, S_{j_b}) = \left\{ \{S_{j_a}, S_{j_b}\} \in B_i \;\middle|\; \begin{array}{l} (t_b(0) \in [0, l_b] \wedge\ t_b(W_R)) \in [0, l_b]) \\ \vee (t_b(l_a) \in [0, l_b] \wedge\ t_b(l_a - W_R)) \in [0, l_b]) \\ \vee (t_a(0) \in [0, l_a] \wedge\ t_a(W_R)) \in [0, l_a]) \\ \vee (t_a(l_b) \in [0, l_a] \wedge\ t_a(l_b - W_R)) \in [0, l_a]) \end{array} \right\}. \tag{3.23}$$

3.6.2 Corridor generation

On every new scan the found traversable segments are linked to the traversable segments in the previous scan where the $\mathcal{F}_o(S_j^k, S_{j_1}^{k-1})$ ((3.23)) is fulfilled.

The desired result is that corridors are formed for each traversable road type visible. An example is shown in Fig. 3.10 where a sidewalk, a bicycle path and a road are visible.

A set of corridors $B_i \in B$ are therefore found by following a unique set of links, as described in (3.24)

$$B_j = \left\{ S_j^k, S_{j_1}^{k-1}, ..., S_{j_n}^{k-n} \right\} \left| \begin{array}{l} \mathcal{F}_o(S_{j_a}^{k-a}, S_{j_{a+1}}^{k-(a+1)}) \\ j_a \in [1..m_a] \\ j_{a+1} \in [1..m_{a+1}] \\ a \in [0..n-1] \\ \wedge\ \text{age}(S^{k-n}) < T_{\text{fade}} \\ \wedge\ |S^{k-n} - P_{\text{robot}}|^s > D_r \end{array} \right. \tag{3.24}$$

where m_a and m_{a+1} are the numbers of traversable segments found in scan a and $a+1$. The links are followed back until the segment has reached the robot position P_{robot} within a (signed) distance $D_r \leq 0.5\,\text{m}$, or the oldest scan has reached a time limit $T_{\text{fade}} = 4.5\,\text{s}$.

There may be more than one possible corridor originating from the same traversable segment S_1^k, eg one segment in scan k could correlate with two segments in scan $k-1$, and further each of these could correlate with multiple segments in the previous scan.

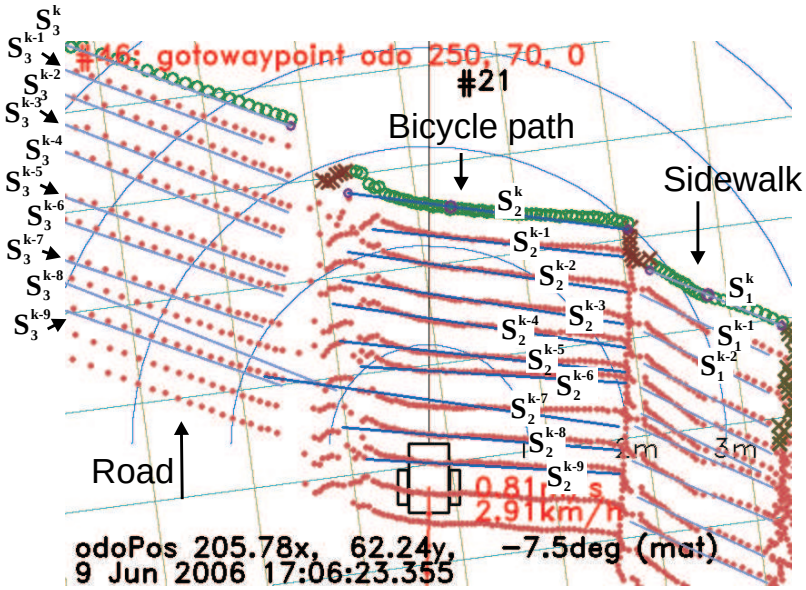


Fig. 3.10: Traversable segments from the most recent 10 scans. The measurements are shown below the lines, as circles (traversable) and crosses (non-traversable) for the most recent scan, and as dots for older scans. The scene is shown in figure 3.9 as seen by the robot camera.

In Fig. 3.10 a corridor is formed for each of the three road types. But the segments S_2^{k-7} and S_3^{k-8} are extra wide and have an overlap that fulfils the criterion, this adds a fourth corridor that switches from the bicycle path to the road at this point.

The situation in Fig. 3.10 will therefore produce the corridors shown in (3.25), where $B_{2,a}$ and $B_{2,b}$ are equal except for the last two scans

$$\begin{aligned} B_{1,a} &= \left\{S_1^k, S_1^{k-1}, S_1^{k-2}, S_1^{k-3}, S_1^{k-4}, S_1^{k-5}, S_1^{k-6}, S_1^{k-7}, S_1^{k-8}, S_1^{k-9}\right\} \\ B_{2,a} &= \left\{S_2^k, S_2^{k-1}, S_2^{k-2}, S_2^{k-3}, S_2^{k-4}, S_2^{k-5}, S_2^{k-6}, S_2^{k-7}, S_2^{k-8}, S_2^{k-9}\right\} \\ B_{2,b} &= \left\{S_2^k, S_2^{k-1}, S_2^{k-2}, S_2^{k-3}, S_2^{k-4}, S_2^{k-5}, S_2^{k-6}, S_2^{k-7}, S_3^{k-8}, S_3^{k-9}\right\} \\ B_{3,a} &= \left\{S_3^k, S_3^{k-1}, S_3^{k-2}, S_3^{k-3}, S_3^{k-4}, S_3^{k-5}, S_3^{k-6}, S_3^{k-7}, S_3^{k-8}, S_3^{k-9}\right\} . \end{aligned} \tag{3.25}$$

When there are multiple corridors from the same segment (as $B_{2,a}$ and $B_{2,b}$ in (3.25)), the best qualified corridor will be maintained only.

The qualification has two parameters: stability in roughness and stability in width. Every traversable segment has a minimum roughness value $\min(R_n) \in S_j$ (see section 3.5.5), the standard deviation σ_B of this value for all the segments in a corridor B generates a roughness quality $Q_r \in]0, 1]$

$$Q_r(B) = (1 + \sigma_B \cdot w_\sigma)^{-1} , \tag{3.26}$$

where $w_\sigma = 1/0.05$ to scale the quality relative to the minimum obstacle size to get a reasonable quality interval.

If the traversable segments start on a smooth road and then switches to a rough bridle path alongside the road, then the corridor with the switch will have a lower quality than a the corridor following either the bridle path all the way or the smooth road all the way.

The road width quality $Q_w \in]0, 1]$ is formed by the standard deviation of the width of the n segments in the corridor

$$Q_w(B) = \left(1 + \overline{w}_B^{-1}\sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(w_B^2) - \overline{w}_B^2}\right)^{-1}, \tag{3.27}$$

where $\overline{w}_B$ is the average width of the corridor B.

The available road corridors are then

$$B_j = \{B_{j,i} \mid \min(Q_r(B_{j,i}) + Q_w(B_{j,i})),\ i \in [1, N]\} \tag{3.28}$$

3.6.3 Road lines

The road lines include the left, the right and the centre road line. These lines are estimated from the laser scanner based road corridor described in the previous section.

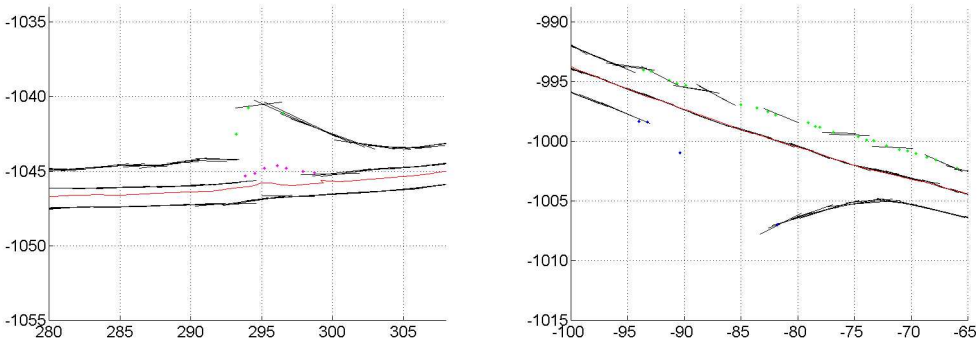
The end point of the segments in a corridor S is fitted to a straight line. This line is taken as the road edge line (L_l or L_r) and the deviation of the segment end points from these lines is used as a measure of the edge detection quality $L_Q \in]0, 1]$ as shown in (3.29)

$$L_Q = (1 + \frac{\sigma_r}{\sigma_{\max}})^{-1}, \tag{3.29}$$

where $\sigma_{\max} = 0.8\,\mathrm{m}$ is a scale value.

Road lines with a road edge quality $L_Q < 0.5$ are discarded, and a quality of $L_Q > 0.8$ indicates a stable, reliable road line. A road edge quality is calculated for the left and the right edge individually. The road edge lines are used as reference when the robot is to follow a specific position on the road. Two examples are shown in Fig. 3.11 where the high quality ($L_Q > 0.8$) road edge lines are shown as solid lines, and edges with less quality ($0.7 < L_Q < 0.8$) are shown as dots.

The centre of the road is often the best place to drive on narrow gravelled roads. The road centre line L_c is defined as the highest point on the road. The highest point is calculated, relative to the traversable segment, as the measurement with the longest signed distance (longer in the z-direction, shorter



(a) (b)

Fig. 3.11: Detected road lines – left, right and road centre. The robot drives from right to left following the thin (red) line in (a) and the centre line in (b). Image (a) shows a narrow asphalt road with a side-road to the right. Image (b) is a gravelled road where the right road edge is hard to detect.

in the x-direction) to the traversable segment S, as shown in (3.30)

$$P_c = \left\{ P_i \in P_{n,m} \;\middle|\; \begin{array}{l} \min\limits^{i\in[n,m]} (AP_{i,x} + BP_{i,y} + C) \\ \text{where} \\ A = v_y,\ B = -v_x,\ C = x_x v_y - x_y v_x \end{array} \right\} \tag{3.30}$$

where $\mathbf{v}$ and $\mathbf{x}$ are segment parameters from (3.21).

Road width

The road width is the average distance between the road edge lines, if these are both valid. If one or both road edges are not valid, then the road width is estimated from the average width of the segments in corridor B. Only the y-coordinate of the measurements is used.

The road width estimate is associated with a quality from (3.27).

3.6.4 Road type

The road type is estimated from the roughness of the traversable segments in the used corridor. The roughness in a traversable segment is relatively homogeneous but may include areas with minor obstacles and may cover parts of the road with some curvature. The roughness is estimated over a road distance of 0.45 m (the robot wheel base) and the roughness used for road type estimation is the minimum roughness R_m over the used traversable segment in the corridor

$$R_m = \min_{n\in S_j}(R_n). \tag{3.31}$$

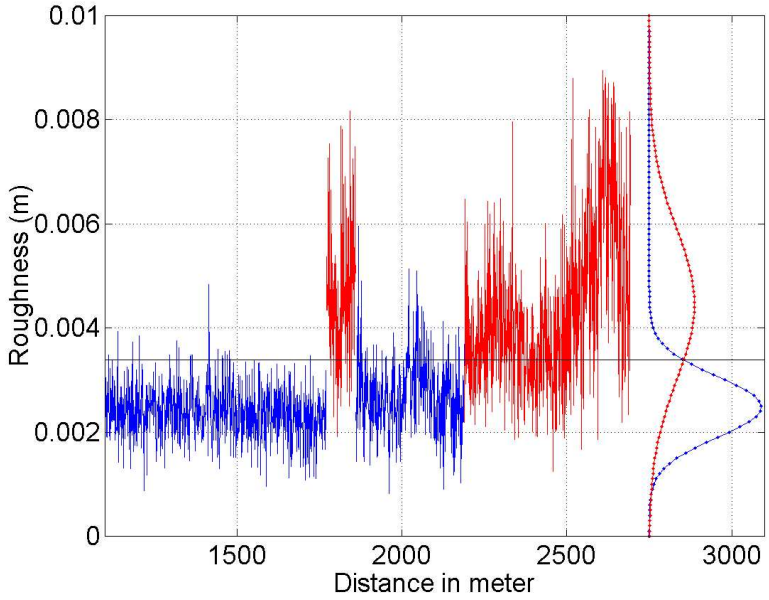


Fig. 3.12: Roughness estimate for the smoothest part of the used traversable segments. From 1100–1770 m and again from 1860–2190 m the road surface is asphalt (blue), the rest has a gravel surface (red). A Gauss distribution with the same mean and variance is shown to the right.

Table 3.2: Roughness estimation of asphalt and gravel road types based on laser scanner measurements.

Road type	Average	Deviation	Unit
Asphalt	$\mu_a = 0.0025$	$\sigma_a = 0.00059$	m
Gravel	$\mu_g = 0.0045$	$\sigma_g = 0.0015$	m

This roughness estimate R_m is shown in Fig. 3.12 for a distance where the robot starts on an asphalt road, crosses a gravelled square, drives on to a narrow asphalt road for some 400 m and then continues on a gravelled road.

The roughness value clearly changes with the road type, but the variation of the roughness is larger than the change in mean value. The laser scanner resolution is 1 mm and the accuracy about 10 mm, so quantisation error and sensor noise are expected to be significant parts of the asphalt roughness especially.

The average roughness and standard deviation of the two road types are for the data in Fig. 3.12 as shown in Table 3.6.4.

Assuming a Gaussian distribution of the roughness for both the asphalt and the gravel road, the distribution is as shown to the right in Fig. 3.12. The point with equal probability for the two road types can then be found by solving

(3.32)

$$
\begin{aligned}
N(\mu_a, \sigma_a) &= N(\mu_g, \sigma_g) \\
\frac{1}{\sqrt{2\pi}\sigma_a} \exp\left(-\frac{(r_{eq}-\mu_a)^2}{2\sigma_a^2}\right) &= \frac{1}{\sqrt{2\pi}\sigma_g} \exp\left(-\frac{(r_{eq}-\mu_g)^2}{2\sigma_g^2}\right) \\
r_{eq} &= 0.0034\,\mathrm{m}\ .
\end{aligned}
\tag{3.32}
$$

Determination of road type clearly requires averaging over some measurements to maintain a stable road type estimate. A cumulative sum (CUSUM) algorithm may be used to determine state change between such two probability distribution functions Blanke et al. (2003), and this algorithm seems applicable for the problem.

The CUSUM algorithm sums the log likelihood ratio for a series of measurements ($r(i)$ for $i \in [1, k]$) of a stochastic variable, as shown in (3.33)

$$
s(k) = \sum_{i=1}^{k} s(r(i)) = \sum_{i=1}^{k} \ln \frac{p_g(r(i))}{p_a(r(i))}\ . \tag{3.33}
$$

This function value $s(k)$ is expected to decrease if the measurement $r(i)$ belongs to the asphalt probability p_a and to increase if it belongs to the gravel probability p_g. When the probability density functions are Gaussian and with equal variance, the expression can be simplified as in (3.34) (from Blanke et al. (2003))

$$
s(k) = \sum_{i=1}^{k} \frac{\mu_g - \mu_a}{\sigma_a^2}(r(i) - r_{eq}). \tag{3.34}
$$

The gravel distribution adapted with the same variance as the asphalt road, positioned symmetrically on the other side of the equal probability line r_{eq} gives an $\mu_g' = r_{eq} + (r_{ex} - \mu_a) = 0.0043$ and $\sigma_g' = \sigma_a = 0.00059$.

When the sum $s(k)$ increases (from a minimum value) by a value h_g the estimate is that the measurements belong to probability function p_g. The value h_g determines the time to detect a change and the false alarm ratio. These values can be determined by the runlength function $L(\mu_s, \sigma_s, h)$ as shown in (3.35) (from Blanke et al. (2003)

Table 3.3: Average 'false alarm rate' and 'time to detect' for the estimation of road type. Both values are in measurement counts. The h value associated with the CUSUM threshold.

Transition	h	false alarm rate	time to detect
asphalt→gravel	7	38700	2.4
gravel→asphalt	10	1700	6.0

$$L(\mu_s, \sigma_s, h) = \left(e^{-2(\frac{\mu_s h}{\sigma_s^2} + \frac{\mu_s}{\sigma_s} 1.166)} - 1 + 2(\frac{\mu_s h}{\sigma_s^2} + \frac{\mu_s}{\sigma_s} 1.166) \right) \frac{\sigma_s^2}{2\mu_s^2} \quad (3.35)$$

where

$$\mu_s = -\frac{(\mu_g - \mu_a)^2}{2\sigma_a^2} \text{ false alarm rate} \quad (3.36)$$

$$\mu_s = \frac{(\mu_g - \mu_a)^2}{2\sigma_a^2} \quad \text{time to detect} \quad (3.37)$$

$$\sigma_s = \left| \frac{\mu_g - \mu_a}{\sigma_a} \right| . \quad (3.38)$$

The same calculations can be performed for the change from gravel to asphalt; there the variance $\sigma_g^2 = 0.0014^2$ is used for both probability functions. The h_g and h_a values have been selected to provide for a reasonable detection time and false alarm rate for both transitions. The result is shown in Table 3.3

With $h = 7$ the average time to detect is 2.5 measurements and the average false alarm rate is one in 38700 measurements, all assuming that the measured values follow the used distribution functions.

The state change from asphalt to gravel is determined from the function in (3.39), following the CUSUM function in (3.34) but is limiting the minimum value to zero. The state change detection from gravel to asphalt is shown in (3.40)

$$\begin{array}{l|l} \text{state:} & \text{state} = \text{asphalt} \\ \text{asphalt} \rightarrow \text{gravel} & b(k) = \max\{0,\, r(k) - r_{eq} + b(k-1)\} \\ & \wedge\ \frac{\mu_g - \mu_a}{\sigma_a^2} b(k) >= h_g \end{array} \quad (3.39)$$

$$\begin{array}{l|l} \text{state:} & \text{state} = \text{gravel} \\ \text{gravel} \rightarrow \text{asphalt} & b(k) = \min\{0,\, r(k) - r_{eq} + b(k-1)\} \\ & \wedge\ \frac{\mu_g - \mu_a}{\sigma_g^2} b(k) <= -h_a \end{array} \quad . \quad (3.40)$$

The resulting road type estimation can be seen in Fig. 3.13. The change

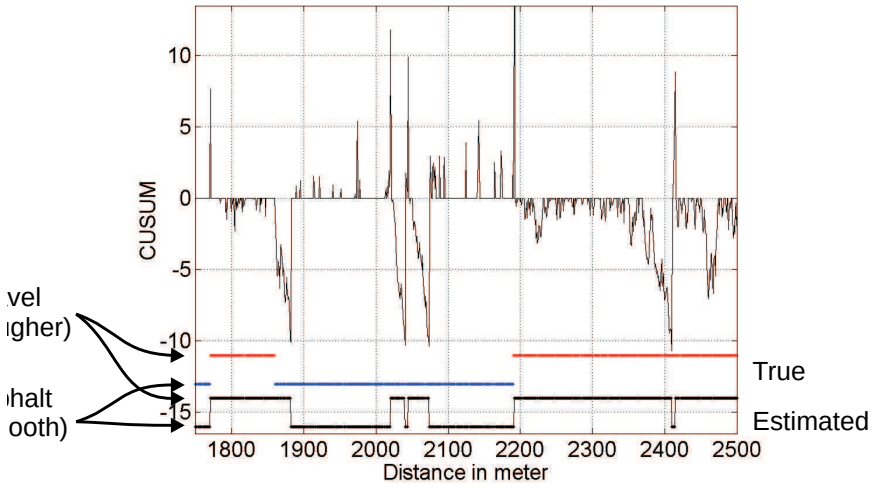


Fig. 3.13: The CUSUM value $s(k)$ used to estimate a change from asphalt (smooth) to gravel (rough) road type. The data is a subset of the data shown in Fig. 3.12. At the bottom is shown the estimated road type compared with the true value.

from asphalt (smooth and homogeneous surface) to gravel (rougher and less homogeneous) is fast, whereas the change back to asphalt takes longer time – as expected. There are two false gravel estimations around the 2050 m distance and at about 2400 m a false asphalt estimate.

The result is not perfect but useful for the purpose, which is to increase edge detection on smooth roads and to allow the road type estimate to be used in behaviour generation.

3.6.5 Results

The classification algorithm has been tested on a set of available paths in the test area. A distance of 3 km has been traversed autonomously using this classification as the main tool to stay on the roads and avoid obstacles.

A few typical classification results are shown in figures 3.6–3.8.

Figure 3.5 shows a smooth asphalt road, with roughness values as shown in Fig. 3.6, where the initial grouping of measurements into homogeneous segments classifies the road in one go.

Figure 3.7 shows an area where the robot is crossing a gravelled road. Here the laser scanner can see only the road surface. Therefore all laser returns should be classified as traversable. The figure shows that the surface is relatively flat. The roughness grouping has divided the area into 6 groups (shown below the measurements in Fig. 3.7), but all of these are recombined into one

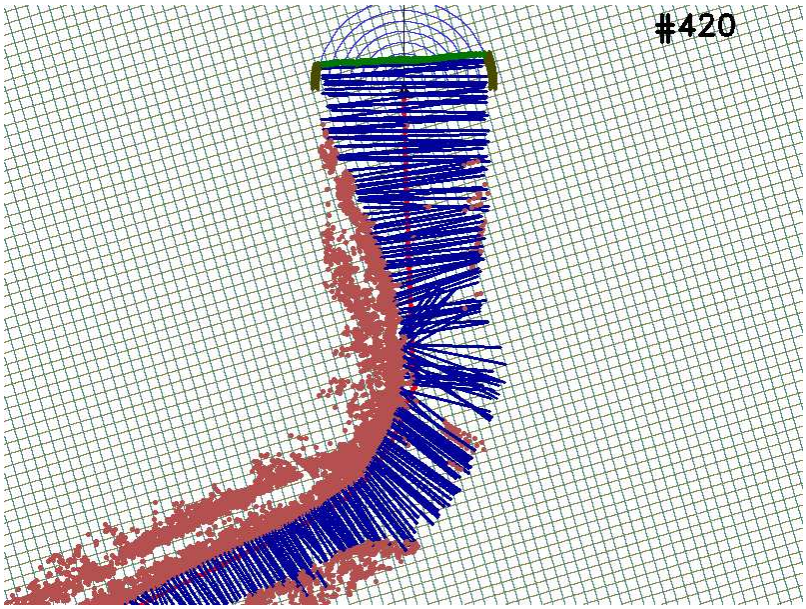


Fig. 3.14: The robot leaves the asphalt road – at bottom left – and enters an open gravelled area, following the left edge at first and then straight towards the exit road. The straight lines (blue) show traversable segments, the (orange) dots and solid area are measurements classified as nontraversable.

traversable segment (shown above the measurements).

Figure 3.8 shows data from a gravelled road with a bridle path crossing. The laser scanner sees the path edges (left and right of the robot in Fig. 3.8) before the bridle path, and sees the bridle path itself as flat areas left and right of the profiled road. The centre of the road is about 15 cm higher than its edges. The algorithm separates the road from the bridle path, and marks both as traversable. The main criterion here, preventing combination of the path with the bridle path, is $\mathcal{F}_c$ (curvature) that prevents concave segments (where the road meets the bridle path) from being combined.

If the transition from the path to the bridle path had been smoother, the segmentation would be more likely to combine the path and the bridle path into one traversable segment.

A longer sequence of road classifications is shown in Fig. 3.14 where the robot leaves an asphalt road and enters a gravelled area in front of the building shown in Fig. 3.15. The segments selected by the navigation process are shown as solid (blue) lines. The laser measurements classified as nontraversable are shown as (orange) dots merging into solid areas.

Table 3.4: The terrain classification, using all extracted features, has been manually evaluated based on representative parts of the autonomous tests. The main road was always classified as traversable in each laser scan; a few percentages of the classifications were too short or too long. On the asphalt road, additional traversable segments were often found in the roadsides left and right, resulting in about three traversable segments in each scan.

Type	road found	too narrow	too wide	trav. segm.
Asphalt road	100%	2.7%	2.0%	3.1
Gravelled area	100%	6%	1%	1.5
Gravelled road	100%	8%	0.7%	1.8

3.6.6 Quality

A representative part of the three main types of the traversable terrain were analyzed manually, to assess the quality of autonomous categorisations. Results are shown in Table 3.4.

On the asphalt road – in Fig. 3.5 and 3.15 – the classification process found the road in 100% of all cases, 2% were too short (shorter than 4.4 m on a 4.8 m wide road), and 2.7% too long (extended into the roadside), but none gave rise to unnecessary manoeuvres. On average about two extra traversable segments were found in each scan, these are mostly found in parts of the grass near the road. These extra segments are however discarded by the navigation layer due to their position and roughness. 300 laser scans were analysed covering about 100 m.

On the gravelled area on the used route (the upper part of Fig. 3.14) a traversable area was always found, but 6% of the scans were too short – defined as less than 4 m from the robot – of which two gave rise to minor unnecessary manoeuvres. 120 laser scans were analysed covering about 40 m.

The gravelled road was about 4 m wide and its edges were rougher than the asphalt edges. The gravelled road could always be found, but sometimes (8%) split into smaller segments, of which two gave rise to minor unnecessary manouvres. 300 laser scans were analysed covering about 100 m.

3.7 Obstacle detection

Laser scanner measurements classified as nontraversable are in general used to form obstacles if they are near a traversable segment. Figure 3.16 shows two examples.

Fig. 3.15: The transition from asphalt road to gravelled area used in Fig. 3.14. The roadsides left and right are often sufficiently smooth to allow formation of additional traversable segments.

An obstacle O^c based on measurement classification is a set of consecutive measurements $P_i \in P_{n,m}$ that fulfils the criteria in (3.41)

$$O^c = \left\{ P_i \in P_{n,m} \;\middle|\; \begin{array}{l} P_i \in \mathcal{C}_n \\ \wedge\, |P_i - P_{i-1}| < d_{\max} \\ \wedge\, d_i < d_t + d_{\text{add}} \end{array} \right\} , \tag{3.41}$$

where $\mathcal{C}_n$ is the class of nontraversable measurements, and $d_{\max} = 0.4\,\text{m}$ is the maximum separation of neighbouring measurements within one obstacle. The laser range d_i of measurement i must be less than the maximum range d_t, of any measurement classified traversable, plus a distance $d_{\text{add}} = 0.5\,\text{m}$.

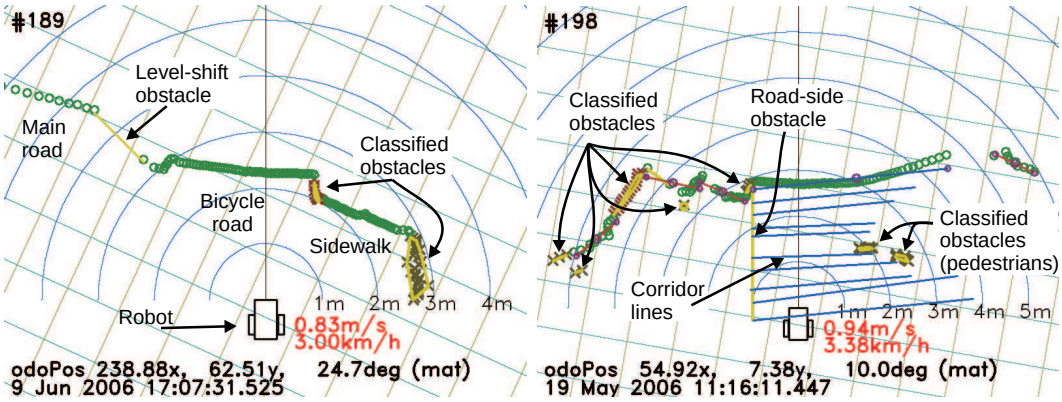
In this way obstacles are generated near traversable areas and at any distance closer to the robot than these. This limits the number of obstacles that need to be handled and includes all the obstacles that need to be avoided in the short term obstacle avoidance behaviour generation.

When a traversable segment extends to a ditch or a negative level shift, there may not be a measurement classified as nontraversable at the edge. Ditch obstacles O^d are therefore added where a traversable segment ends in a ditch as defined in (3.42)

$$O^d = \left\{ P_{i-1,i} \;\middle|\; \begin{array}{l} (P_i \in \mathcal{C}_t \;\wedge\; d_{i-1} - d_i > d_{\text{ditch}}) \\ \vee (P_{i-1} \in \mathcal{C}_t \;\wedge\; d_i - d_{i-1} > d_{\text{ditch}}) \end{array} \right\} , \tag{3.42}$$

where $\mathcal{C}_t$ is the class of traversable measurements and $d_{\text{ditch}} = 0.4\,\text{m}$ is the minimum range separation to declare a ditch or level shift.

The edges of traversable segments are not automatically classified as obstacles, as the full extend of the traversable segment may be hidden behind an obstacle or be out of sight due to the curvature of the road. Road edges are though, when they are detected with a sufficiently high quality, added as O^L obstacles as defined in (3.43). This prevents the behaviour planning from



(a) (b)

Fig. 3.16: Detection of obstacles. The laser measurements classified as non-traversable are divided into obstacle polygons. Additionally if a traversable segment ends with a level shift to a lower level the positions around the level shift are marked as obstacles as shown in (a) where the level goes down to the main road. In (b) the road edge has a high quality (the residual from the line fitting is low) and is included as an obstacle.

crossing road boundaries even if the area on the other side of the road boundary is classified as traversable (deliberate crossing of a road boundary is therefore not supported)

$$O^L = \{L \in \{L_r, L_l\} \,|L_Q > Q_{\min}\} \ , \tag{3.43}$$

where $Q_{\min} = 0.9$ is the required quality limit for addition of road edge lines as obstacles.

The measurements in an obstacle are reduced to a convex polygon. This reduces the amount of data needed to describe the obstacle and simplifies calculations. Further, the amount of vertices needed to describe the obstacle is reduced, so that close vertices are removed (reduction limit set to 0.03 m).

The obstacles will be further processed when merged with obstacles from previous scans as described in chapter 5.

3.7.1 Wall detect

When the robot drives directly towards a wall the first detection of the wall will look like a flat road surface – only slightly elevated. When the robot moves the elevation increases until the height threshold $h_{\text{sup}} = 0.2$ is passed, the wall is now about 1.3 m from the robot.

A wall detection function is therefore added that invalidates a corridor if it ends in a wall. The wall test compares the position and orientation of two

consecutive traversable segments while the robot has moved (at least 0.05 m). The two segments must be within 0.04 m of each other and their orientation must be within 2.3°.

When the wall in this way is detected twice (using three consecutive scans) the corridor is assumed to end in a wall, and thus invalidated as a traversable corridor.

In a rough outdoor environment it happens that the road detection gets closer – due to robot tilt – at the same speed as the robot movement, there is thus a possibility that the corridor is mistaken to be a wall. A few instances of this type have been recorded, typically resulting in an erroneous stop.

If a corridor flagged to be a wall continues to correlate without further triggering of the wall detector, the wall flagging is removed after four non-wall scans.

3.8 Summary

This chapter has described a comprehensive set of features extracted from the laser scanner measurements. These should form a solid base for obstacle avoidance and behaviour generation. The features extracted include:

Traversable segments describing parts of the surface that is evaluated to be traversable. Each segment is further qualified with a roughness estimate.

Traversable corridors generated from a set of correlated traversable segments. The corridors describe a part of the road – primarily in front of the robot – that is traversable; there may be multiple of such corridors, eg describing a road fork or multiple lanes originating from eg a road and a sidewalk.

Road lines are generated as a by-product of the corridors. The road lines are calculated from the left, right and centre of the corridor. A quality value is associated with each line to allow discrimination of lines influenced by obstacles or partially out of laser scanner range.

Road type estimate based on roughness. The tested road types include asphalt (smooth) and gravelled (rough) roads, and the road type is estimated reasonably well.

Obstacles describing areas that should be avoided by the behaviour generation function.

Requirements for laser scanner positioning and scan rate are further evaluated to allow sufficient obstacle detection ability and sufficient warning time for manoeuvring.

The limitations of the laser scanner are primarily the limited range (the road is detected at a distance of about 2.6 m in front of the robot) and in the limited ability to discriminate a road from its sides in the cases where the sides are rather flat and smooth.

3.9 Further improvements

Further improvements are expected to be obtainable especially in the following areas:

Road line detection is not continued from one scan to the next, but calculated anew using the most recent corridor only. A method of estimating the road edge continuously using eg a Kalman filter is expected to produce superior results and may enable continued road edge detection when passing obstacles. Such road edges would be valuable, especially if the robot was to perform mapping.

A Roll and pitch sensors would allow for slightly better estimation of road curvature, and this could allow for better qualification of a number of the used thresholds in the classification process.

Obstacle detection at times detect obstacles near maximum range in open areas or when passing a side road, this could probably be avoided by usage of some of the partial results from the filtering.

Chapter 4

Vision based perception

4.1 Introduction

The objective for any sensor is to extract useful information.

A vision sensor has the potential to extract any visible feature inside the camera view, but especially two limitations make vision-based sensors difficult to use for navigation purposes:

A camera sensor is an angle-only sensor. This first limitation necessitates that the sensor must have additional information to be able to deliver 3D positions of the features. This additional information could be knowledge of the size of a feature, the distance from camera to the feature, or angle information from a sensor at another position, eg another vision sensor.

The second limitation is the use of illumination of opportunity. An image of a stationary scene looks different in different illuminations, the shadows change, the colour changes with the colour of the illumination, bright illumination of some features and week illumination of objects in the shadows make it difficult for the camera to get sufficient detail in both situations.

When the illumination is controlled and the scene has known features, eg an orange ball of a known size on a horizontal green surface, then the detection is simplified, but parts of the ball may not seem orange as part of the surface – the highlight area – just reflects the illumination and thus has the same colour as the illumination.

In an outdoor environment the illumination is changing and the size of the seen objects is mostly unknown. The use of vision for navigation in such an environment poses a significant challenge. This chapter presents vision-based solutions and discusses the obtained results.

4.2 Related work

The use of cameras to aid robot navigation has been described in a number of papers using a large number of different approaches.

Stereo vision is one of the fundamental methods to create a 3D model of the angle-only measurements from the camera. The basic method is to recognise the same features in two or more images, and from accurate knowledge of the camera position and orientation to calculate the triangulation points to reconstruct the distance to the feature. One project described in Loaïza et al. (1999) and Loaïza et al. (2001) uses two cameras mounted on top of each other and does stereoscopic calculation on contrast straight lines found in the images, the Loaïza et al. (2001) project puts emphasis in colour on each side of the extracted lines to improve correlation. The project Se et al. (2001) uses three cameras in a right angle triangular configuration to remove false correlations. The project also extracts a number of (scale invariant) features before stereo calculation and then stores these points (about 3000 for one room) for the ongoing navigation.

A camera with fish-eye lenses that cover 360° around and above the robot can be very useful for localisation, especially in an indoor environment. Shah & Aggarwal (1997) use this configuration to find significant lines ending in geometric vanishing points and from these and vertical lines build a model of the traversed corridor.

In vision solutions Bertozzi & Broggi (1997) argue that this problem can be divided into two subproblems: lane following and obstacle detection, and describe a stereo-vision based solution for both. The lane following depend on the bright lane markings and a stereo-based solution to detect the angle to moving obstacles (cars).

Edge detection in vision systems is one of the possibilities to identify road borders and had some success already in 1986, as described in Wallace et al. (1986).

A method for road following using vision and neural network to estimate the main road direction was developed by Jochem et al. (1993) for lane following, and in Jochem et al. (1995) expanded to detect side roads and 'Y' junctions.

A simple method – with the same basic principle as used in this project – is described in Horswill (1994), where the edges of a homogenous structureless area in front of the robot is taken as obstacle free area for navigation.

Lane following using single camera solution is also described in Liatsis et al. (2003), where edges are detected and tracked from one frame to the next using edge magnitude and orientation in the area of the image, where the road edge is expected.

The DARPA Grand Challenge 2004 race demonstrated great difficulties in road following and obstacle avoidance for autonomous vehicles as described in

eg Behringer et al. (2005).

4.3 Limitations and possibilities

Vision depends on ambient light and provides a 2D colour and texture view of parts of the surroundings.

The image colour depends on the illumination (eg sunlight and shadows), the time of year (eg trees are green in the summer, red or yellow in the fall and partially transparent in the winter) and weather conditions (eg roads are often black when it rains, grey when they are dry and possibly white during the winter). The camera may lack dynamic range to detect colours in both sunny and shaded parts of an image. Limited illumination at night and weather conditions like fog, rain or snow will further limit the results of vision-based solutions.

Some of these factors may have little or no influence on some vision sensors or solutions may be available to avoid or compensate for the effects.

A vision sensor is an angle-only sensor, and the 3D reconstruction of the scene therefore requires additional information, typically in the form of multiple images from different locations, assistance from other sensors, or assumptions related to the features seen in the image.

The possibilities of using vision are – despite the limitations mentioned above – almost endles. Evolution has proved the success of vision, as large groups of the living creatures on earth use vision as one of their main sensors.

The solutions can be divided into two groups, one group uses images from more locations to place the objects seen in a 3D environment eg stereovision and structure from motion; the other group makes assumptions on the seen features, being objects of a known size or an assumption of the layout of the scene, eg roads are predominantly flat and has mostly parallel edges.

Stereovision was initially evaluated for this project, but the tested solution was temporarily discarded due to experienced problems during robot movement (the solution is described in appendix A). Stereovision units for robot navigation are becoming commercially available and may in time replace the need for laser scanner based obstacle detection – especially at short ranges.

For this project the near obstacle detection and the road detection are handled by a laser scanner. The prime purpose for a vision system is therefore to assist where the laser scanner comes short, ie at longer ranges and where there are ambiguities in the laser scanner detections.

The two vision sensor methods presented here are both based on presumptions of the viewed features to reconstruct a 3D world model: a road outline sensor that assumes the road seen by the camera to be flat and in the same plane as the robot base and a guidemark recognition solution that requires the

physical size of the recognized artificial guidemark to be known in advance. The first solution should supplement the laser scanner sensor and the second uniquely identify the robot position, once a guidemark is recognised.

4.4 Road outline

The assumption is that it is possible to estimate the outline of the road by analysing the image, based on an appropriate seed area. Such a seed area may be provided by the laser scanner.

The main features describing the road are its homogeneity. But there may be variation in the visual expression due to eg shadows, sunlight, specular reflections, surface granularity, flaws, partially wet or dry surface and minor obstacles like leaves on the road.

The road detection is therefore based on two features: the chromaticity $\mathbf{C}$ and the intensity gradient ∇I.

The chromaticity is colour stripped from intensity as shown in (4.1) based on a RGB (Red, Green, Blue) image

$$\mathbf{c} = \begin{bmatrix} c_{\text{red}} \\ c_{\text{green}} \end{bmatrix} = \begin{bmatrix} r/(r+g+b) \\ g/(r+g+b) \end{bmatrix} . \tag{4.1}$$

Each pixel position $H_{i,j}$ is classifiable into class $\mathcal{R}$ ={road, not road}. The $\mathcal{R}_{\text{road}}$ classification is defined as in (4.2) and the $\mathcal{R}_{\text{not road}} = \complement\mathcal{R}_{\text{road}}$ holds the remaining pixels

$$\mathcal{R}_{\text{road}}(H_{i,j}) = \left\{ H_{i,j} \;\middle|\; \begin{array}{l} (1-K_{\text{bal}})P_c(\mathbf{C}(H_{i,j}))+ \\ K_{\text{bal}}P_e(\nabla I(H_{i,j})) \\ > K_{\text{lim}} \end{array} \right\} , \tag{4.2}$$

where $K_{\text{bal}} = 0.43$ gives the balance between chromaticity and edge sensitivity, and $K_{\text{lim}} = 0.65$ the detection sensitivity. Both values are found experimentally. A sample filtering is shown in Fig. 4.1 for each of the two components as well as combined.

$P_c(\cdot)$ is a probability function based on the Mahalanobi distance of the chromaticity relative to the seed area

$$P_c(i,j) = \left(1 + w_c(\mathbf{c}'_{i,j} - \overline{\mathbf{c}})^T \mathbf{Q}^{-1} (\mathbf{c}'_{i,j} - \overline{\mathbf{c}})\right)^{-1} , \tag{4.3}$$

where $\mathbf{Q}$ is the chromaticity covariance for the seed area, $\overline{\mathbf{c}}$ is the average chromaticity in the seed area, and $w_c = 0.022$ ensures appropriate numerical values for the result.

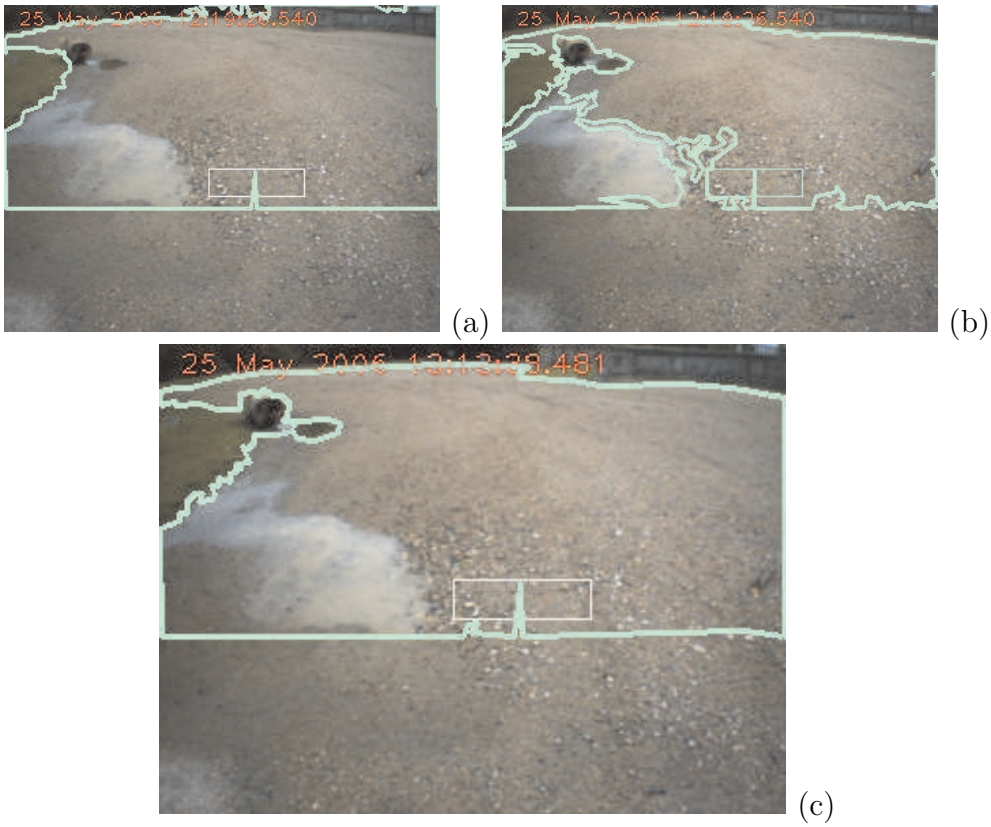


Fig. 4.1: Road outline extraction based on chromaticity (a), on gradient detection (b) and combined (c). In the top left corner there is a stone fence; this is not distinguished from the gravel road surface using the chromaticity filter in (a). The gradient filter (b) makes a border to the pit (bottom left). The combined filter (c) outlines the traversable area as desired. The seed area classified by the laser scanner is shown as a (thin) rectangle. The part of the image below the seed area is not analysed.

$P_e(\cdot)$ is a function based on the intensity gradient $\nabla I(H_{i,j})$

$$P_e(i,j) = \left(1 + w_e \left[\left|\frac{\partial I(i,j)}{\partial i}\right|^2 + \left|\frac{\partial I(i,j)}{\partial j}\right|^2\right]\right)^{-1}, \tag{4.4}$$

where $w_e = 0.067$ is a weight factor (the image intensity has the value range [0,255]).

The intensity gradient is calculated using a Sobel operator. The Sobel kernel size is selected as appropriate for the position in the image, ie the lower in the image (closer and more detailed) the larger the kernel (5×5 pixels at the bottom and 3×3 at the top for the used 320×240 image resolution).

The pixels at the road contour are evaluated only, ie from the seed area

pixels are tested towards the image edge or road border, the road border is then followed (clockwise) back to the seed area. This is a very computationally efficient analysis method as only a subset of the pixels in the image are analysed and no filtering is needed on the full image. A disadvantage is that small obstacles (lower than the camera height) in the middle of the road may remain undetected, when all surrounding pixels indicate traversable road. This disadvantage could be avoided by testing pixels inside the road outline at appropriate intervals.

4.4.1 Shadows

Shadows are a problem as the colour information in the shade is more bluish from the blue sky whereas the colour in the sun is more reddish – and thus may not match the seed area. In full sunshine and full shadow the automatic white balance of the camera will compensate for this, but when both are present in the same image it needs to be considered. The white balance in sun and shadows influences the balance between red and blue chromaticity values only, whereas the green is relatively unaffected. The intensity in the shade is lower than in the sun, and the intensity is therefore a first choice for a correlated value that could be used to compensate for the shift in white balance.

In Fig. 4.2(b) the blue and red chromaticity is shown as a function of intensity for a part of the road area in the image shown in Fig. 4.2(a) with mixed shade and sun (on the asphalt road part of the image).

The figure shows that darker areas have an increased blue and decreased red component – as expected. The difference in colour balance is a function of the used camera and the colour of the sky, but the tendency shown in Fig. 4.2 is taken to be representative for this camera. The chromaticity value uses the red and green values only, therefore the red chromaticity value c_{red} is adjusted as defined in (4.5)

$$\mathbf{c}' = \begin{bmatrix} c_{\text{red}} + (\overline{I} - I) g_{\text{red}} \\ c_{\text{green}} \end{bmatrix}, \tag{4.5}$$

where $I = (r + g + b)$ is the intensity, $\overline{I} = 1.5 \times 256$ is the average intensity (the linearization point) and $g_{\text{red}} = 0.066$ is the linearized gradient of the red chromaticity as a function of intensity – derived from Fig. 4.2.

This shade-compensated $\mathbf{c}'$ is used as the basis for calculation of $\overline{\mathbf{c}}$ and the covariance matrix $\mathbf{Q}$.

The seed area is taken from the most recent laser scan, and the traversable segment with the least roughness is used. The seed area is taken as the central third of this segment; this will in most cases avoid the areas near the road borders, which may not be representative for the road. This central part is projected into the image and used as a seed area.

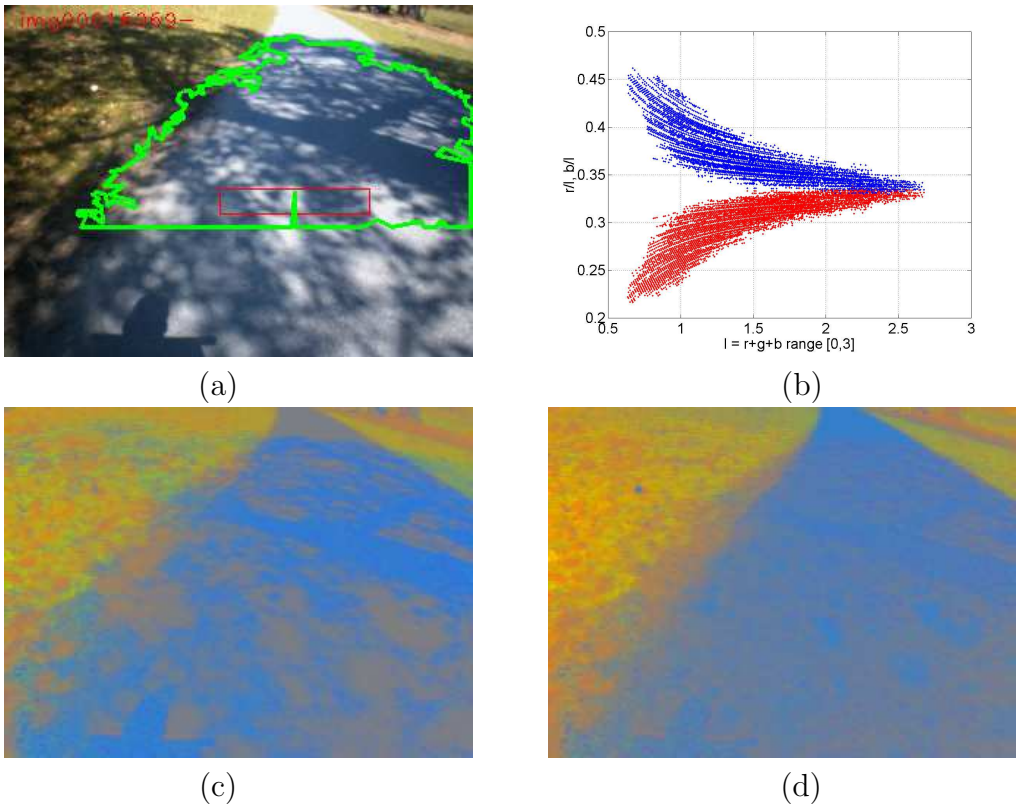


Fig. 4.2: A part of the mixed shade and sunshine on the asphalt road in (a) is analysed for white balance in (b). The x-axis is intensity and the y-axis is blue (upper) and red (lower) chromaticity. The curves show that the darker areas are more bluish than the brighter areas. The chromaticity of (a) is shown in (c), here the shadows are clearly more blue than the areas in the sun. Image (d) shows the effect of shadow compensation, the effect of shadows is reduced significantly (both on the road and in the grass area).

4.4.2 Road-outline polygon

The road outline is found as a sequence of neighbouring pixels that fulfils the road criteria in (4.2). From the seed area (centre top) a limitation line is drawn down to the bottom of the image, and further a limitation line is drawn across the image below the seed area to limit the polygon size (the area closer to the robot than the seed area is covered by the laser scanner).

The series of pixels found describe a polygon. The number of vertices in this polygon is reduced using a modified version of a method described by Douglas & Peucker (1973). The method finds the most distant vertex from the edge line between a start vertex and an end vertex. The most distant vertex is added and the search is continued until the distance to the most distant vertex is less than a given threshold. The accuracy needed for the road-outline polygon

changes with the position in the image: one pixel covers a larger road area at the top of the image than at the bottom, on the other hand pixels at the bottom describe positions that are closer to the robot and are thus more relevant for obstacle avoidance. The threshold used is one pixel at the top of the image and 4.5 pixels at the bottom with a linear scaling between these values.

4.4.3 Projection to robot plane

The pixel coordinates in the reduced polygon needs to be projected to the robot plane to be comparable with the laser scanner solution. This is done in three steps: compensation for radial lens error, finding a vector in the pixel direction and finally finding the position where this vector crosses the robot base plane.

- The pixel position $\mathbf{p}^e = [p_x^e, p_y^e]^T$ found in the image is adjusted for radial lens error to $\mathbf{p} = [p_x, p_y]^T$ using the two major radial error parameters K_1 and K_2 for the lens (values are found in section 2.4.5). The adjustment is performed as in (4.6) – from Carstensen (2001)

$$\begin{aligned} r^2 &= (\mathbf{p}^e - \mathbf{h})^T(\mathbf{p}^e - \mathbf{h}) \\ d_r &= K_1 r^2 + K_2 r^4 \\ \mathbf{p} &= \mathbf{p}^e + (\mathbf{p}^e - \mathbf{h}) d_r \ , \end{aligned} \tag{4.6}$$

  where $\mathbf{h} = [h_x, h_y]$ is the head point in the image (the geometric image centre is used).

- A 3D line $\mathbf{a} = [\mathbf{x}_a + t\mathbf{v}_a]$ from the projection centre in the direction of the pixel coordinate $\mathbf{p}$ is found (in camera-oriented coordinates) $\mathbf{a}^c = [\mathbf{0} + t\mathbf{v}_a^c]$ as in (4.7)

$$\mathbf{v}_a^c = \begin{bmatrix} v_x^c \\ v_y^c \\ v_z^c \end{bmatrix} = \begin{bmatrix} 1 \\ -(p_x - h_x)/c \\ -(p_y - h_y)/c \end{bmatrix} \ , \tag{4.7}$$

  where c is the focal length, and the vector end position is taken at an arbitrary fixed x-distance of 1 m.

  The line origin and orientation vector are then coordinate converted to robot-oriented coordinates as in (4.8)

$$\begin{aligned} \mathbf{a} &= [\mathbf{x}_c + t(\mathbf{v}_a)] \\ \mathbf{v}_a &= \mathbf{R_c}\mathbf{v}_a^c \ , \end{aligned} \tag{4.8}$$

  where $\mathbf{x}_c == [x_x, x_y, x_z] = [0.43, 0, 0.86]$ is the camera position, $\mathbf{R_c} = \mathbf{R}_\kappa \mathbf{R}_\varphi \mathbf{R}_\Omega$ is the camera rotation matrix where primarily $\mathbf{R}_\varphi$ has a value different from unity, as the camera is rotated along the y-axis by $\varphi = 22^\circ$ as from section 2.4.5.

- The line from the camera to the pixel position is crossing the robot plane in the searched position **b** (unless parallel with robot plane), and **b** is found as in (4.9) utilising that the robot plane is the x, y plane

$$\begin{aligned} \mathbf{b} &= [\mathbf{x}_c + t_b \mathbf{v}_a] \\ t_b &= -\frac{x_z \, |\mathbf{v}_a|}{v_z} \ , \end{aligned} \tag{4.9}$$

  where $\mathbf{v}_a = [v_x, v_y, v_z]$.

  If v_z is positive or zero then there is no crossing (at least not in front of the robot), such results are replaced with a position close to the horizon in order to get a proper polygon in robot-oriented coordinates. Positions above the horizon of the robot plane may be classified as road when the robot tilt (or roll) places parts of the real horizon above the robot plane or if objects in the horizon (erroneously) are taken as road.

The road-outline polygon is now in robot-oriented coordinates and is ready for use by the navigation and obstacle avoidance functions.

4.4.4 Road outline results

The road width estimate and the road width stability can be taken as a performance measure of the vision and laser scanner sensors. The method is tested primarily on a 3 km route in the test area. The navigation is guided by a script specifying how to follow the roads and for how long. Parts of this route are analysed.

An analysis is shown in Fig. 4.3 for an asphalt road and in Fig. 4.4 for a gravelled road. The weather conditions were overcast with mild showers. The road width is estimated in front of the robot based on the available data at the time of manoeuvre decision. The vision-based road width is taken as the average distance between the road lines, where both road lines are inside the image (see next chapter for more details).

The road width data in Fig. 4.3 covers a distance of about 500 m, and shows a stable road detection both by the laser scanner sensor and by the vision senor, the vision with a slightly higher uncertainty, but with good correlation between the two sensors. The road-outline estimate in Fig. 4.3(b) shows an almost perfect correlation with the road edges. There is a slight tendency that the vision estimate shows a narrower road width than the laser scanner.

The gravelled road in Fig. 4.4 shows data from a distance of about 300 m with a side road about halfway. The road width seen by the vision sensor is clearly narrower than the width from the laser scanner. The reason is twofold: the vision estimate is too narrow and the laser scanner estimate is too wide.

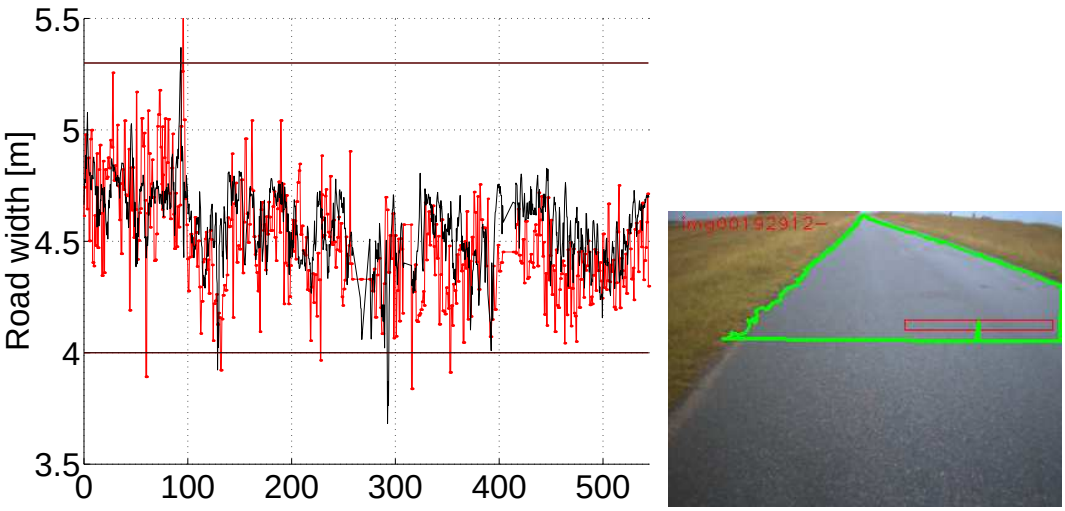


Fig. 4.3: Road width estimation (a) based on laser scanner (*solid black*) and vision-based (*red with dots*). A 500 m section of an about 5 m wide asphalt road. The vision-based outline for the first part of the road is shown in (b).

The width seen by the vision system is projected to a plane at the robot base, and as the road profile is convex, the true road width is slightly wider. Typically the road is 10 cm higher at the centre, and with a true road width of 3.5 m (as it is most of the way), the projection would show a road width of 3.13 m. This corresponds nicely with the measured value.

The laser scanner estimated a road width of typically 3.5 to 4 m with only little correlation with the vision-based road width. The reason is mostly that the road roughness is not much lower than the roughness of the near part of the roadsides. Figure 4.5 shows a set of measurements from the early part of the data series in Fig. 4.4. The blue lines (in Fig. 4.5(a)) between the robot and the laser scanner measurements are the traversable segments used for calculating the road width. Especially the right roadside shows variations in the segment width, none is too short, but some are too wide. The thin green lines show the estimated road lines. This effect is mostly significant where the roadside is in the same plane as the road (especially in the early part of the data series in Fig. 4.4).

Seasonal variation

An about 600 m section of the route are shown in Fig. 4.6(a) and (c) taken in the spring (early May) in sunshine and in Fig. 4.6(b) and (d) the same road section in midsummer (late July) during an overcast weather condition. The first part of the section is a narrow asphalt road (up to about 300 m) and then

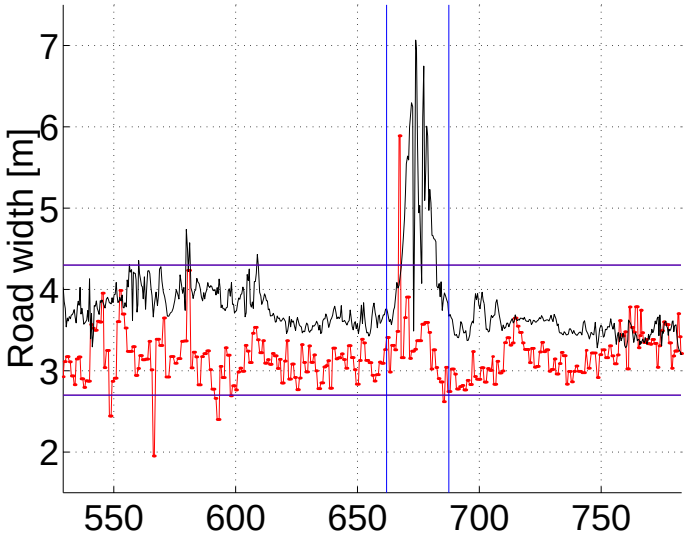


Fig. 4.4: Road width estimation based on laser scanner (solid black) and vision-based (red). A section from an about 3.5 m wide gravelled road (a). At about 675 m a side road makes the road estimate wider.

a gravelled road for the remaining part. Two side roads are passed at about 100 m and 500 m.

The width of the asphalt road (about 3 m) is detected equally well in spring and in summer. The gravelled road appears wider in early spring than mid-summer, and the uncertainty seems much higher, both for the laser scanner based width and the vision-based width.

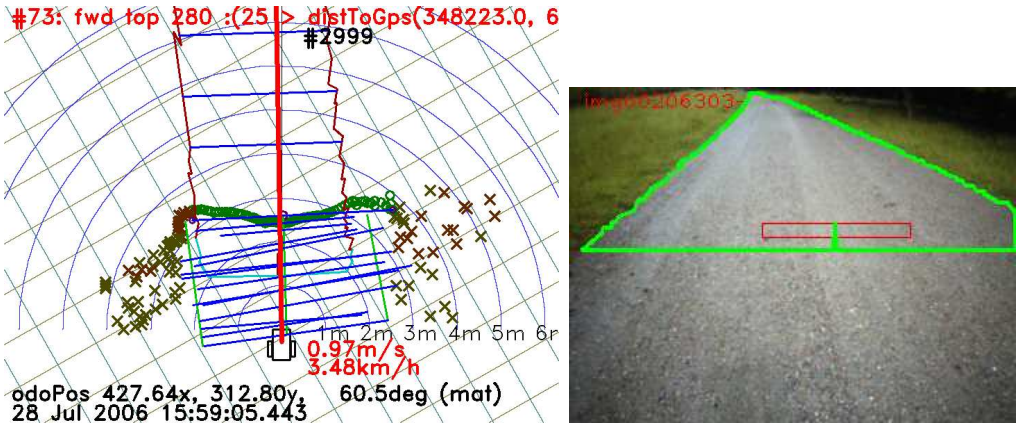
The laser scanner estimate is wider as the grass is still relative flat after a long winter, so the roughness difference between the road and the roadside is marginal. In midsummer the grass is higher and the border thus easier to distinguish.

For the vision sensor the partially gray grass (from the winter) is much harder to distinguish from the road than at midsummer. An example can be seen in Fig. 4.7(h), where especially the right roadside is problematic.

Another issue is sunshine and shadows. Below open trees in sunshine, like in Fig. 4.7(c) and in Fig. 4.6(c) – both from the early part of the data series – the road width estimation is more difficult resulting in a more insecure estimate (visible at about 50 m in Fig. 4.6(a)).

Summary

The road width estimates are summarised in table 4.1 for the laser scanner and vision sensor, respectively. The first two rows are from the same test but at different road types – asphalt and gravel, respectively. The last two rows in



(a) (b)

Fig. 4.5: A set of measurements at the early part of the data series used in Fig. 4.4. Both laser scanner data and the vision road outline is shown in (a). The vision-based outline is shown in (b).

Table 4.1 are from a narrow asphalt road segment that is partially below large trees and here the road outline estimates failed in 16% of the measurements on a day with sunshine, compared to just 2% under gray and wet weather condition. Failed measurements are counted as a too wide or too narrow estimate ($> 20\%$ of true value).

The conclusion is that the vision sensor performs reasonably well, and sometimes better than the laser scanner. The performance is slightly better on asphalt roads than on gravelled roads. Gray grass and scattered shadows reduce the performance.

Shadows

Most road outline situations are handled reasonably well, the shadows from overhanging trees in Fig. 4.7(a), (b) and (c) and from the lightpole in (d) produce usable results. The partially wet asphalt in (e) is handled. The tiled sidewalk in (f) is only partially solved, especially the sunny spots in a rather dark image as in (c) and (f) are difficult. The gravel road in (h) and (i) are basically fine, but the sharp shadow in (i) is taken as an obstacle. In (h) the side road to the left is partially found, but part of the right roadside is taken as road. Painted road markings are normally taken as obstacles.

All the shown images in Fig. 4.7 are produced with the same algorithm and the same weight and threshold settings.

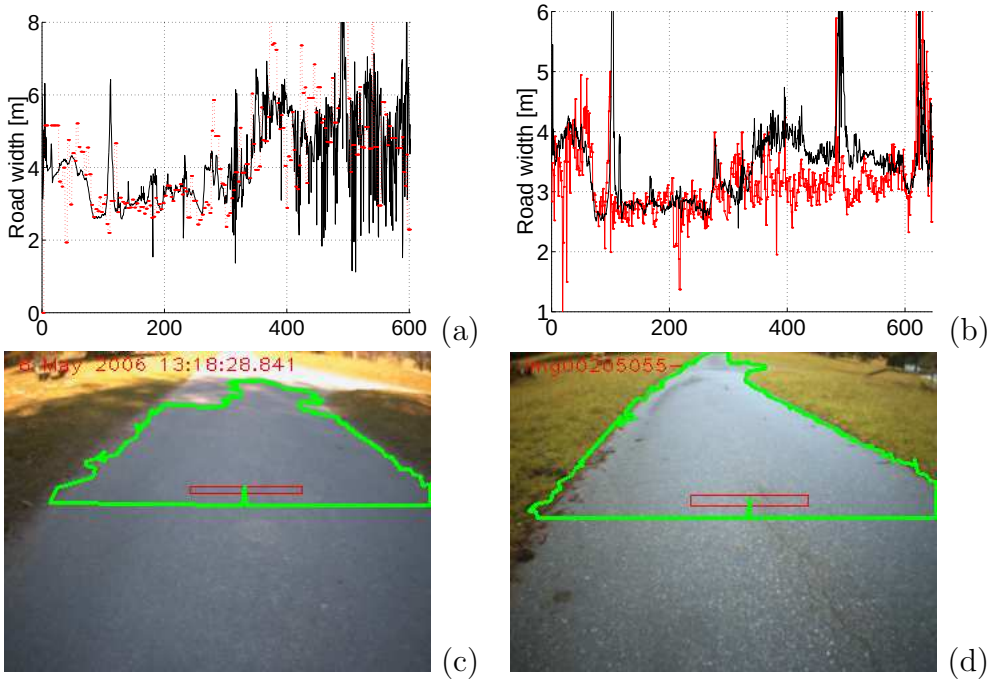


Fig. 4.6: Road width data from a 3–4 m narrow asphalt up to 320 m where the road surface changes to gravel; there are side roads at 100 m and 500 m. Figures (a) and (c) are from early may in sunshine with shadows from the overhanging trees; whereas (b) and (d) are from late july in gray/wet weather. The image (c) from just before the side road at 100 m showing a transition from shade to sun, and that the detected road outline covers the part in the shade only. The image (d) from about the same spot three months later. The vision data is affected by the sunshine to shade transitions, resulting in too narrow road width estimates at places (especially at around 50 m).

Road outline limitations

The seed area may include part of roadside if the laser scanner classifier fails to separate a flat roadside from the road. In these cases the vision sensor fails a proper solution.

Small obstacles in the middle of the road with insufficient height to cross the visual road outline will not be detected. The unseen obstacle could be of the same height as the camera and would remain unseen if the outline search finds visual evidence of the road left, right, above and below the obstacle in the image. Such obstacles could potentially remain undetected until seen by the laser scanner.

Road coloured obstacles or part of obstacles that visually are too similar to the road will remain undetected.

Table 4.1: Road width estimate summary from the data shown in Fig. 4.4 and Fig. 4.3 is in the first two rows. The last two rows are from the narrow asphalt road segment in the early part of Fig. 4.6 both in the early spring, the first under overcast (wet) weather and the last under clear sunshine. The 'w/n' column is the too wide or too narrow count as a percentage of the measurement count N.

Road	True	Laser based			Vision based				
segment	width	mean	σ	N	mean	σ	w/n	N	ref
Asphalt	4.9	4.7	0.17	1200	4.5	0.24	1%	474	4.3
Gravelled	3.5	3.7	0.48	600	3.2	0.32	3%	245	4.4
Asphalt	3–4	3.5	0.63	890	2.8	0.36	2%	224	wet
Asphalt	3–4	3.3	0.46	482	2.8	0.53	16%	79	sun

Road paintings will typically be detected as obstacles (not part of the road).

Sharp shadows may trigger the edge detector sufficiently to be taken as an obstacle.

Change in road surface type, eg change from asphalt to gravel, may limit the road outline.

Sensor saturation may occur, especially in sunny spots in an otherwise dominantly shady area.

4.5 Guidemark recognition

Landmark navigation is based on the assumption that the robot from recognising a landmark can get a localisation reference. The landmark could be artificial and placed to be recognised by the robot, ie for indoor applications a method is to place unique landmarks on the ceiling and let a robot camera look for these landmarks, and further place the landmarks at so short intervals that the robot could navigate from one landmark to the next with sufficient accuracy to be able to find the next guidemark.

The guidemark itself could be at a known position, or just act as a unique reference position so that any ambiguity or accumulating errors could be resolved or reduced when recognising the guidemark.

The initial position of the robot could be resolved by recognition of a unique artificial landmark. This landmark could refer to an entry in the robot database with knowledge of that specific area. One or more landmarks could be placed close to a charging station that may require accurate navigation.

For these purposes a camera-based landmark system has been designed to explore the possibilities.

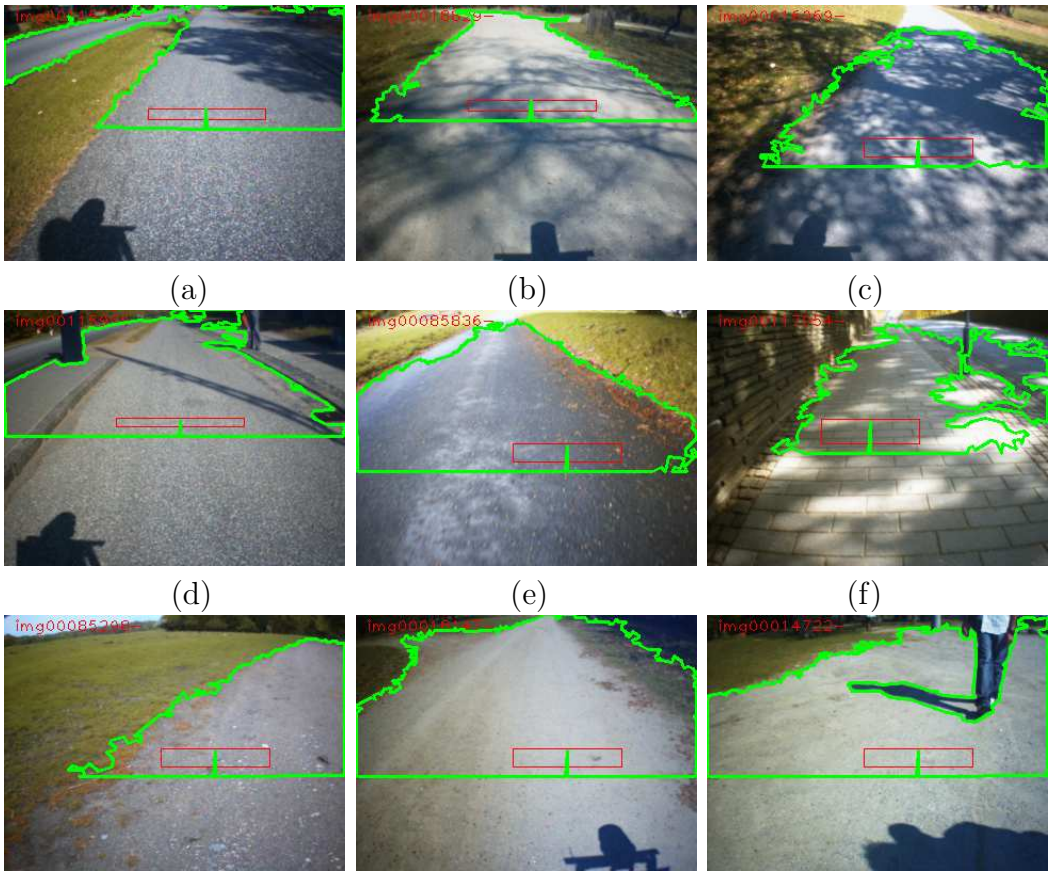


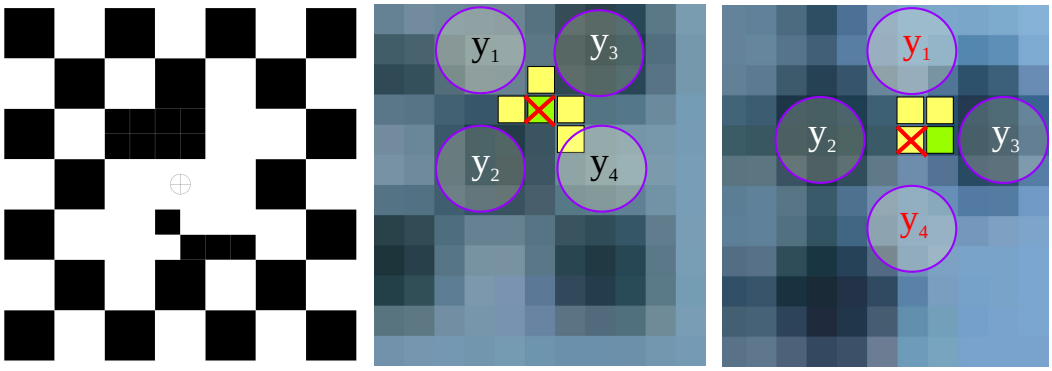
Fig. 4.7: Examples of the described road outline detector, the green line is the found outline and the red square is the seed area.

4.5.1 The landmark

The landmark must be detectable at different distances and at different angles. One of the easy and stable detectable shapes are checkerboard corners, they are scale invariant, and to some extent viewing angle invariant. The guidemark should be simple to reproduce, and thus printable on a sheet of paper would be preferable.

The final design was selected as shown in Fig. 4.8(a), with a double checkerboard frame and a central area for a unique code.

The centre code holds 9 square areas; each can be filled with a 4 bit code. Two of the top left squares are used as orientation marks, leaving 7 for codes. A few code combinations can be confused with the orientation mark and must thus be avoided, leaving some 24 usable bits except for 2^{16} unusable code combinations, or in total 16711680 code possibilities. A smaller frame with just 4 squares in the centre would be sufficient in most cases, with one corner as



(a) (b) (c)

Fig. 4.8: The guidemark in consist of a checkerboard frame and a central code. The camera view of the lower left part is shown in (b) and (c). The corner filter uses four 3×3 areas to detect a corner at the centre position.

orientation mark, and at maximum 3 bits used in the remaining a total of 9 bits or 512 codes would be available.

The guidemark in Fig. 4.8(a) has two black bits set in each of the two least significant squares – bottom right inside the frame – corresponding to the value $9C_{\text{hex}}$ or 156 decimal.

The guidemark is detected in four steps: corner detection, frame detection, code detection and frame localisation, each of these steps are described in the following.

4.5.2 Corner detection

The corner detection is done by performing a comparison of four areas in the image expected to match two black areas and two white areas.

The full image is first filtered using a 3×3 pixel Gaussian kernel, summing the pixels inside the kernel area – in a $G(i, j)$ function – to the centre pixel position (i, j) (eg the y_1 area in Fig. 4.8(b)). The summation uses a Gaussian distribution for weights with a $\sigma = 0.95$. This yields the central pixel a weight of about as much as the sum of weights of the remaining 8 pixels.

A set of corner pixels $\mathcal{C}_1$ at pixel positions $\mathbf{a} = [r, c] \in \mathcal{C}_1$ are found in the image as defined in (4.10)

$$\mathcal{C}_1 = \left\{ \begin{array}{c} \mathbf{a} = [r, c], w \\ r, c \in \text{image} \\ w = y_1 + y_4 - y_2 - y_3 \end{array} \middle| \begin{array}{l} y_1 - y_2 > k_c \\ \wedge\, y_1 - y_3 > k_c \\ \wedge\, y_4 - y_2 > k_c \\ \wedge\, y_4 - y_3 > k_c \end{array} \right\}, \tag{4.10}$$

where

$$y_1 = G(r-2, c-2) \tag{4.11}$$
$$y_2 = G(r+2, c-2) \tag{4.12}$$
$$y_3 = G(r-2, c+2) \tag{4.13}$$
$$y_4 = G(r+2, c+2) \tag{4.14}$$
$$k_c = \frac{1}{3}[\max(y_1, y_2, y_3, y_4) - \min(y_1, y_2, y_3, y_4)] + k_I \; . \tag{4.15}$$

An intensity difference is thus required from all bright corners to all black corners. This difference k_c is increased with the intensity difference from the brightest to the darkest block as in (4.15) and includes a fixed minimum threshold $k_I = 7$ (the intensity coding is 255 for white and 0 for black). This ensures that a guidemark in both bright areas and darker areas are detectable.

The filter will detect a corner that is bright in the upper-left and lower-right part. Intensity-reversed corners are found in the same way by exchanging the calculation of $\{y_1, y_4\}$ with $\{y_2, y_3\}$ in (4.11) to (4.14) for the second corner set $\mathcal{C}_2$.

The guidemark may however be observed at different orientations depending on the positioning of the guidemark and the camera. By adding a guard band of one pixel between the 4 corner areas, the filter will be relatively insensitive to viewing angle rotation of the guidemark relative to the camera, the corner detection sensitivity will be reduced as the angle increases, and at 45° the sensitivity will be zero.

To be able to detect guidemarks at any rotation angle a second set of filter masks (rotated 45°) is added as shown in Fig. 4.8(c), using the same filter function as in (4.10), except that the relative positioning of the four areas y_1, y_2, y_3, y_4, is replaced with (4.16) to (4.19) producing a third corner set $\mathcal{C}_3$

$$y_1 = G(r-3, c) \tag{4.16}$$
$$y_2 = G(r, c-3) \tag{4.17}$$
$$y_3 = G(r, c+3) \tag{4.18}$$
$$y_4 = G(r+3, c) \; . \tag{4.19}$$

Again replacing the calculation of $\{y_1, y_4\}$ with $\{y_2, y_3\}$ in (4.16) to (4.19) yields the fourth corner set $\mathcal{C}_4$.

The distance between the four summed areas is in the 45° version 4.2 pixels and in the not rotated version 4 pixels. When the block size in the image is less than four pixels (or 4.2 when rotated) the detection probability will decrease. A not rotated guidemark can thus be detected at slightly longer distances compared to a rotated guidemark.

The corner pixels in $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4\}$ are then enumerated individually using 8-connectivity into the total set of corner groups H_n. The centre of each corner group $\mathbf{h}_n(H_n)$ is found as shown in (4.20)

$$\mathbf{h}_n(H_n) \quad = \quad \frac{1}{\sum_{i \in H_n}(w_i)} \sum_{i \in H_n} (\mathbf{a}_i w_i) \; , \tag{4.20}$$

where the accuracy improvement by using the intensity weight w_i derived in (4.10) is not quantified nor optimised.

In Fig. 4.8(b) and (c) the detected corner pixels are shown as bright squares in yellow and green, where green marks the pixel closest to found corner position.

When a guidemark is rotated approximately 22.5° it happens that the same corner is detected by two filters, ie one of $\{\mathcal{C}_1, \mathcal{C}_2\}$ and one of $\{\mathcal{C}_3, \mathcal{C}_4\}$, such corners are combined to one if the distance between the centres $\mathbf{h}_a - \mathbf{h}_b <$ 2.9 pixels.

4.5.3 Frame detection

The corner positions have to match the frame of the guidemark, ie from a frame corner there should be two sets of 6 corners each describing a straight line following a frame edge, and further the corners should have almost the same separation. Figure 4.9 shows an example with three guidemarks – rotated at different angles. The original image is shown faintly in the background. The corner pixels are colour coded, each colour corresponds to one of the four filters. The pixel closest to the detected corner is shown in a darker colour.

From all corner positions the up to eight closest neighbours are found within a maximum distance – of $\frac{1}{5}$ image height – allowing a guidemark to fill the whole image. A frame edge corner set $\mathbf{f}_j$ is therefore described as six corners on an approximately straight line fulfilling the requirements in (4.21)

$$\mathbf{f}_j = \left\{ \begin{array}{l} \{\mathbf{h}_i\} \\ i \in [1, 2, .., 6] \\ \mathbf{h}_i \in \mathbf{h}_n \end{array} \middle| \begin{array}{l} \mathbf{v}_2 = \mathbf{h}_2 - \mathbf{h}_1 \\ \hat{\mathbf{h}}_2 = \mathbf{h}_2 \\ \hat{\mathbf{h}}_{i+1} = \hat{\mathbf{h}}_i + \mathbf{v}_i \\ \mathbf{v}_{i+1} = \mathbf{v}_i + k_g(\mathbf{h}_i - \hat{\mathbf{h}}_i) \\ |\mathbf{h}_i - \hat{\mathbf{h}}_i| < k_{\lim}|\mathbf{v}_i| \\ i \in [2, 5] \end{array} \right\} \; , \tag{4.21}$$

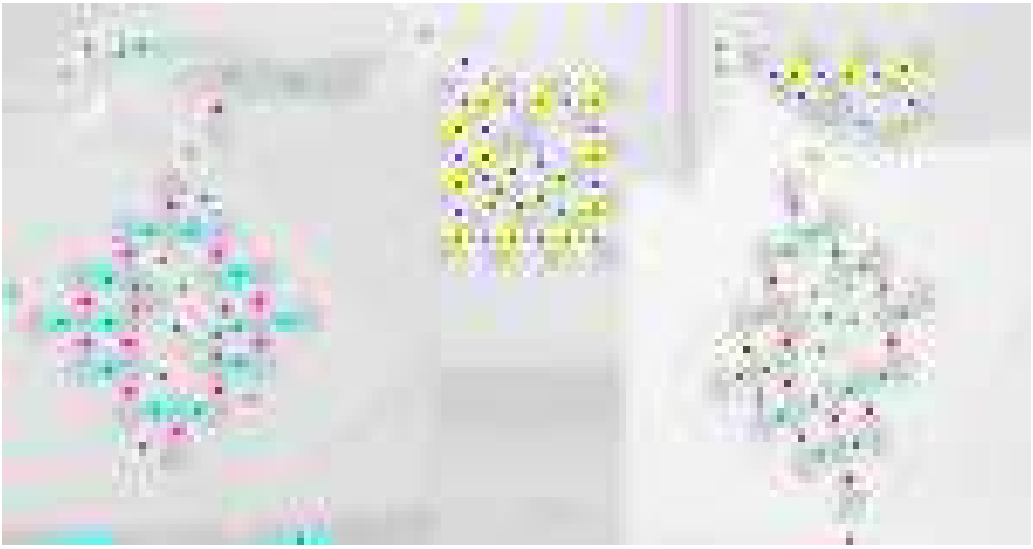


Fig. 4.9: The corner pixels from three guidemarks are shown. The different colours correspond to the four used corner filters.

where $k_{\text{lim}} = 0.37$ and $k_g = 0.5$.

A frame corner should be the end point of at least two frame edges. In this way a full frame $\mathbf{F}$ is a set of four frame edges as described in (4.22) with the corners in each edge ordered as described in frame conditions

$$\mathbf{F} = \left\{ \{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3, \mathbf{f}_4\} \;\middle|\; \begin{array}{l} \mathbf{F}_{n,m} = \mathbf{h}_m \in \mathbf{f}_n \\ \mathbf{F}_{1,1} = \mathbf{F}_{2,1} \\ \mathbf{F}_{2,6} = \mathbf{F}_{3,1} \\ \mathbf{F}_{1,6} = \mathbf{F}_{4,1} \\ \mathbf{v}_n = \mathbf{F}_{n,6} - \mathbf{F}_{n,1} \\ |\mathbf{v}_1 - \mathbf{v}_3| < k_p |\mathbf{v}_1| \\ |\mathbf{v}_2 - \mathbf{v}_4| < k_p |\mathbf{v}_2| \end{array} \right\}, \tag{4.22}$$

where the frame is described counter-clockwise, so that $\mathbf{f}_1$ is the topmost edge and $\mathbf{f}_2$ is the leftmost edge. The edges of a frame should in pairs be approximately parallel and of equal lengths, ie $\mathbf{f}_1 \parallel \mathbf{f}_3$ and $\mathbf{f}_2 \parallel \mathbf{f}_4$ with the limits $k_p = 0.5$.

The six corners $\mathbf{h}_1$ to $\mathbf{h}_6$ are fitted to a straight line, and the crossing of this line with the line from one of the adjacent edges is used as the true frame corner. The frame can then alternatively be described by these four frame corners as defined in (4.23), describing the frame counter-clockwise with $\mathbf{h}_1'$ as

the topmost corner

$$\mathbf{F} = \left\{ \mathbf{h}_1^{'}, \mathbf{h}_2^{'}, \mathbf{h}_3^{'}, \mathbf{h}_4^{'} \right\} \ . \tag{4.23}$$

4.5.4 Code detection

The code in the guidemark requires detection of black and white areas in the centre of the frame, and each of the code bits cover an area of only a quarter of the blocks in the frame. The intensity level that separates a black bit from a white bit must further be determined.

The size of the code bits are selected so that the probability of detection for the frame and the code vanishes at about the same distance. At the distance where the frame is detected with a 95% probability, the code is correctly detected with a probability of about 95% (of the instances where the frame is detected).

A frame grid is constructed by dividing the distance between two adjacent corners on every frame edge in two. The corners are projected to the fitted line edge, but the distance between the corners are not equalised. The correct distance between the corners may change over the frame if the guidemark is seen in perspective, but this effect is mostly significant if part of the guidemark is very close to the camera, and being close to the camera the code detection is usually easy, as a high number of pixels are available for each code bit. At longer distances all cells in the grid will be very close to the same size, and here the grid accuracy is more important for code recognition. A minor improvement in code detection may therefore be obtainable if the grid spacing along the edges was made equal.

Figure 4.10 shows an example of some detected guidemarks. The code grid is painted in green and includes the inner part of the frame blocks as well as the code area itself. All pixels inside the green area are evaluated as belonging to one of the cells, and the average intensity of the pixels inside the cell is used to estimate its value.

Half of the cells covering the frame blocks are always white, the other half black. The average intensity for these cells is used as a threshold value when classifying bits as black or white in the code area. The histograms in the bottomleft corner of Fig. 4.10 show the distribution of intensity values for each of the three detected guidemarks. Left is black, right is white on the histogram line. The histogram colour should match the grid colour painted on top of the guidemark.

The two all black code areas for the orientation mark are located, and the code values in the remaining code area are ordered accordingly. The two FF_{hex} squares mark the top left corner of the code area, and in this orientation the four bits in each block are coded as follows:

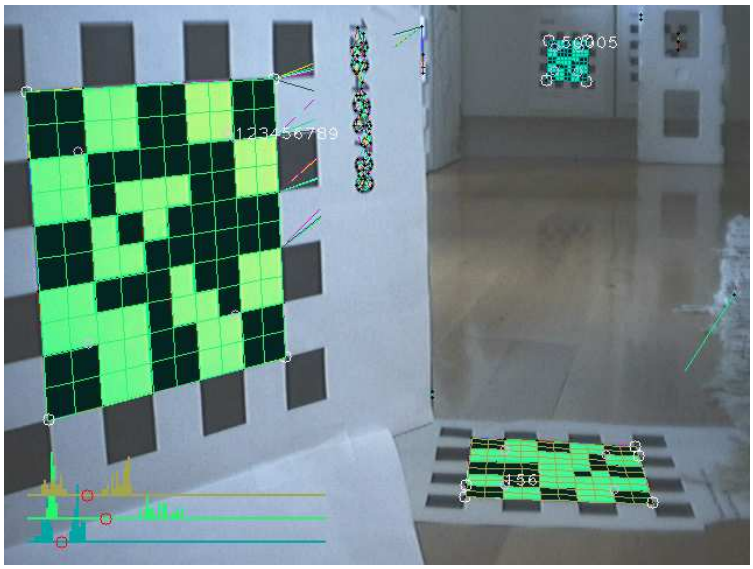


Fig. 4.10: Three guidemarks are visible in the same scene to demonstrate the limitations. All guidemarks have a frame width of 17.5 cm. The near is at 0.46 m and the far at 3.2 m, the one on the floor is tilted 79° – all relative to the camera position. The image resolution is 640 × 480 pixels.

1	2
4	8

The ordering of the code blocks from the most significant to the least significant is located as follows:

*	*	1
2	3	4
5	6	7

(4.24)

The code in the large guidemark in Fig. 4.10 is therefore $75BCD15_{\text{hex}}$, or in decimal 123456789.

4.5.5 Guidemark position

The guidemark position relative to the camera can be estimated when the size of the guidemark is known and the camera geometry is available.

The guidemark corner positions are known with a relatively high accuracy, as these are averaged from the line fitting of the frame edge. The perspective described by the positioning of the guidemark corners should therefore allow a reasonably accurate estimate of the orientation too.

The guidemark position and orientation is estimated using least square parameter estimation. From the frame extraction algorithm above the position in the image of the frame corners are known $\mathbf{h}'_i = (h_r, h_c)$. The position of the corners on the guidemark surface is known from the guidemark design as four coordinate pairs.

The coordinates on the guidemark is selected as being seen in the same way as a robot, that is x being forward (in front of the guidemark), z being up, and when looking in the direction of x (from behind the guidemark) y is to the left. When looking at the guidemark on a wall, then z is up and y is right.

The centre of the guidemark is taken as the reference position, ie the top right frame corner has the frame coordinate $\mathbf{B} = [0, b_y, b_z]^T$ with positive values for both b_y and b_z.

A guidemark may be at any position $\mathbf{g}_t = [x, y, z]^T$ relative to the robot and rotated following the normal convention: first turned κ around the vertical z axis with positive being counter-clockwise, then tilted Φ around the y axis with positive being a down tilt and finally roll Ω around the x axis with positive being a roll to the right.

When a point on the guidemark surface $\mathbf{B} = [0, b_y, b_z]^T$ is being seen at the 3D position $\mathbf{A} = [a_x, a_y, a_z, 1]^T$ in robot coordinates, then $\mathbf{A}$ and $\mathbf{B}$ are related with the guidemarks orientation and position $(x, y, z, \Omega, \Phi, \kappa)$ (also in robot coordinates) as in (4.25)

$$\begin{bmatrix} 0 \\ b_y w \\ b_z w \\ w \end{bmatrix} = \mathbf{R}_\Omega \mathbf{R}_\Phi \mathbf{R}_\kappa \mathbf{T} \mathbf{A} \ , \tag{4.25}$$

where $\mathbf{R}_\Omega$, $\mathbf{R}_\Phi$ and $\mathbf{R}_\kappa$ are rotation matrices in homogeneous coordinates and $\mathbf{T}$ is a translation matrix as shown below:

$$\mathbf{R}_\Omega = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\Omega) & \sin(\Omega) & 0 \\ 0 & -\sin(\Omega) & \cos(\Omega) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.26}$$

$$\mathbf{R}_\Phi = \begin{bmatrix} \cos(\Phi) & 0 & -\sin(\Phi) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\Phi) & 0 & \cos(\Omega) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.27}$$

$$\mathbf{R}_\kappa = \begin{bmatrix} \cos(\kappa) & \sin(\kappa) & 0 & 0 \\ -\sin(\kappa) & \cos(\kappa) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.28}$$

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{4.29}$$

The conversion between image coordinates and the 3D position A of the point on the guidemark are – except for lens distortion – as defined in (4.30)

$$\begin{bmatrix} h_r w \\ h_c w \\ w \end{bmatrix} = \begin{bmatrix} -1 & 0 & h_x \\ 0 & 1 & h_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 1/c & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \\ 1 \end{bmatrix} \tag{4.30}$$
$$\mathbf{I} = \mathbf{bPA} ,$$

where $\mathbf{I} = [h_r w, h_c w, w]^T$ holds the row h_r and column h_c of the corresponding pixel position of $\mathbf{A} = [a_x, a_y, a_z, 1]^T$ in the image. The $\mathbf{b}$ matrix offsets the position to get positive row and column values by adding the (optical) image centre (h_x, h_y) and changing the direction of the row axis to get down as positive. The $\mathbf{P}$ matrix adds the perspective by scaling the row and column values by $1/c$ into w proportional to the distance from the camera a_x. The a_y direction corresponds to columns in the image with changed sign, and the height a_z corresponds to image rows.

When the camera is positioned at the centre of the robot coordinates, then the two equations (4.30) and (4.25) can be combined as shown in (4.31)

$$\begin{bmatrix} h_r w \\ h_c w \\ w \end{bmatrix} = \mathbf{bPT}^{-1} \mathbf{R}_\kappa^T \mathbf{R}_\Phi^T \mathbf{R}_\Omega^T \begin{bmatrix} 0 \\ b_y \\ b_z \\ 1 \end{bmatrix} . \tag{4.31}$$

The right side of this equation can be evaluated to three functions of the unknown $\mathbf{v} = [x, y, z, \Omega, \Phi, \kappa]^T$ and the known position (b_y, b_z) as in (4.32)

$$\begin{bmatrix} h_r w \\ h_c w \\ w \end{bmatrix} = \begin{bmatrix} f_r(x, y, z, \Omega, \Phi, \kappa, b_y, b_z) \\ f_c(x, y, z, \Omega, \Phi, \kappa, b_y, b_z) \\ f_w(x, y, z, \Omega, \Phi, \kappa, b_y, b_z) \end{bmatrix} . \tag{4.32}$$

The last w equation can be inserted into the first two as in (4.33) where the six unknowns are replaced by the vector $\mathbf{v}$

$$\begin{bmatrix} h_r f_w(\mathbf{v}, b_y, b_z) - f_r(\mathbf{v}, b_y, b_z) \\ h_c f_w(\mathbf{v}, b_y, b_z) - f_c(\mathbf{v}, b_y, b_z) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} . \tag{4.33}$$

To solve for the six unknowns at least six equations are needed, so the four corners of the guidemark frame yield eight equations by substituting b_y, b_z, h_r and h_c in (4.33) with the values from the remaining three corners. The problem may then be solvable. The eight functions on the left side of (4.33) should all evaluate to zero with the correct value of the six unknowns

$$\mathbf{F} = \mathbf{0} \; . \tag{4.34}$$

As the functions are nonlinear the six unknown parameters are estimated using Newton's iteration method.

With an initial guess of the vector $\hat{\mathbf{v}}$ the equations will (probably) not be zero, but assuming that the errors are small and the equations are approximately linear at the guessed position, the error can be compensated for by a linear adjustment $\Delta\mathbf{v}$ as shown in (4.35)

$$\mathbf{F}(\hat{\mathbf{v}}) + \mathbf{J}(\hat{\mathbf{v}})\Delta\mathbf{v} = \mathbf{0} \; . \tag{4.35}$$

where $\mathbf{F}(\hat{\mathbf{v}})$ is the value of the eight equations evaluated with the guessed set of parameters $\hat{\mathbf{v}}$ and $\mathbf{J}(\hat{\mathbf{v}})$ is the Jacobian of $\mathbf{F}$ with respect to the unknowns in $\mathbf{v}$ taken at $\hat{\mathbf{v}}$, finally $\Delta\mathbf{v}$ is the adjustment to the guess needed to get the required zero result.

A better guess of $\mathbf{v}$ would therefore be $\hat{\mathbf{v}}_2$ as shown in (4.36)

$$\hat{\mathbf{v}}_2 = \hat{\mathbf{v}} + \Delta\mathbf{v} \; . \tag{4.36}$$

The estimated adjustment $\Delta\mathbf{v}$ is found by solving (4.35) as:

$$\Delta\mathbf{v} = -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{F} \; . \tag{4.37}$$

Equations (4.35), (4.36) and (4.37) are then repeated, setting $\hat{\mathbf{v}} = \hat{\mathbf{v}}_2$ for the next iteration, until the estimated parameters have converged sufficiently.

The pixel position in the image is adjusted for radial lens error (as in (4.6)) prior to insertion into the functions in $\mathbf{F}$.

The iteration is terminated when the parameter change $\Delta\mathbf{v}_n$ in iteration n is significantly small according to the stop criteria in (4.38)

$$\text{stop criteria} = \left| \begin{array}{l} \left[\hat{x}, \hat{y}, \hat{z}, \hat{\Omega}, \hat{\Phi}, \hat{\kappa}\right] \\ |\,[\hat{x}, \hat{y}, \hat{z}]\,| < P_{\text{inf}} \wedge \\ \left|\left[\hat{\Omega}, \hat{\Phi}, \hat{\kappa}\right]\right| < R_{\text{inf}} \end{array} \right. \; . \tag{4.38}$$

When looking at a guidemark that is tilted slightly forward or backward it may be difficult to see the difference, this could indicate local optimum that could trap the parameter estimation. Figure 4.11 shows the pixel error as a function of a combination of turn (κ) and tilt (Φ). This shows the correct value

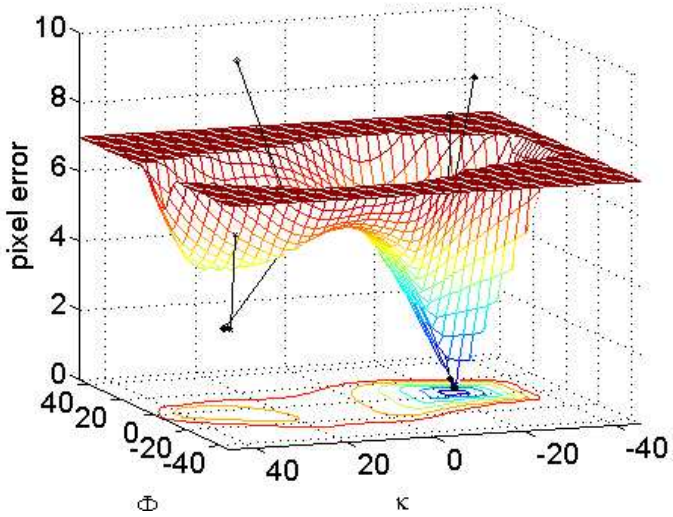


Fig. 4.11: Parameter estimation for the position and orientation of a guidemark has (often) local minimum. The pixel error is shown as a function of turn (κ) and tilt (Φ) of the guidemark (limited to a maximum error of seven pixels). The remaining parameters (x, y, z, Ω) are kept at the correct value.

for these parameters ($\Phi = 5°$) and ($\kappa = -22°$), but also a local minimum at about ($\Phi = -13°$) and ($\kappa = 35°$).

The rotation (Ω) of the guidemark will have four equally accurate solutions, as there is no discrimination of the four frame corners. But the position of the code index is known and is used to get the correct Ω value.

The position has a local minimum at the same distance behind the camera, but this is easily avoided by selecting an initial guess in front of the camera (a positive x value).

To avoid the (κ, Φ) local minimum, four initial positions in the four quadrants in the κ, Φ coordinate system are tested, and after a few iterations the parameter set with the least pixel error is continued to get a final estimate. The iteration error progress is shown in Fig. 4.11 as four black lines, of which two ends in the local minimum at $\Phi = -13°$ and $\kappa = 35°$ with a minimum pixel error of 2.5 pixels, compared to 0.12 pixels at the global minimum in $\Phi = 5.0°$ and $\kappa = -22.8°$.

4.6 Guidemark results

The estimation accuracy of a guidemark position is dependent on the viewing angle of the guidemark as shown in Table 4.2.

The position estimation error in (x, y) is about 0.2 cm and is partially cor-

Table 4.2: Relative estimation accuracy of a sequence of 100 position requests of a guidemark at 2.2 m at different turn angles of the guidemark. All source images are of size 640 × 320 pixels, the guidemark has a frame block size of 2.5 cm, and each frame block is covered by at least the stated number of pixels. The used USB camera has a focal length of 1050 pixels.

Viewing	Position	Orientation			Block	N
angle κ	$\sigma_{\|x,y\|}$	σ_Ω (roll)	σ_Φ (tilt)	σ_κ (turn)	pixels	samples
0°	1.7 mm	0.04°	1.55°	0.61°	11.9	100
10°	1.3 mm	0.06°	0.72°	0.27°	11.7	100
30°	2.2 mm	0.12°	0.21°	0.12°	10.3	100
60°	2.5 mm	0.10°	0.11°	0.06°	5.9	24

related with an estimation error in the guidemark orientation; typically a small tilt combined with a slight turn makes the guidemark seem slightly smaller and thus further away. When the turn angle is zero (guidemark is facing the camera) the relative estimation error in roll (σ_Ω) is uncorrelated with the other errors and thus small, at larger turn angles the roll error increases and the error value is now correlated with the other estimated parameters.

The obtainable absolute position accuracy is dependent on the mounting accuracy of the camera, the focal length of the lens and the accuracy of the estimated lens (radial) errors. With the used USB camera an absolute position accuracy of less than 5 cm and an angle accuracy of less than 5° are obtained within the camera coverage area.

When a guidemark is viewed with a rotation of 22.5° – just in between the two sets of corner filters ($\mathcal{C}_{1,2}$ and $\mathcal{C}_{3,4}$) – the sensitivity is slightly reduced. The effect is a reduction of the distance at which the guidemark can be detected.

The number of pixels needed for each of the squares in the frame to be able to detect the guidemark is shown in table 4.3 as 'block pixels'.

Table 4.3: The number of pixels needed for each frame block to detect guidemarks at different rotation angles relative to camera. The distance in metre is for a camera with a focal length of 525 pixels and an image size of 320 × 240 pixels.

Orientation	pd = 0.5		pd = 0.95	
of grid	pixels	metre	pixels	metre
0°	3.8	3.4	3.9	3.3
22.5°	4.6	2.8	4.8	2.7
45°	4.2	3.1	4.3	3.0

When the probability of detection (pd) is about 0.95 the code is evaluated correctly with a probability of about 0.95 too (for the detected guidemarks).

Stable guidemark detection requires that each of the blocks in the guidemark frame should be covered by at least five pixels. When the guidemark is not at the distance with the optimal focus the detection distance will decrease further.

The used camera has a configurable contour enhancement feature intended to compensate for limited bandwidth in an analogue video signal and to enhance the image sharpness visually. This feature is turned off to improve the guidemark detection capability.

4.7 Summary

Two vision sensors are presented in this chapter: a road outline sensor and a guidemark sensor.

The road outline sensor is able to extend a sample area to cover the visible area with the same properties as the sample area. The visual appearance of a road changes with the illumination and the surface type, but the presented sensor that combines a colour (chromaticity) analysis with edge detection is able to detect the road outline in many surface type and illumination combinations: gravelled roads, wet and dry asphalt roads with a range of sunshine and shadow illumination. The method is fast, simple and efficient as long as the seed area is representative for the road.

The sensor fails if the seed area is not representative for the road and additionally very sharp shadows are excluded from the road outline, and thus both false positives (non-road included) and false negatives (road excluded) may occur. For these reasons the sensor is not recommendable as a stand alone sensor for detection of obstacles and traversable terrain. The sensor is however an excellent supplement to the laser scanner based road detector described in chapter 3, and provides supplementary road classification, obstacle and junction detection beyond the laser scanner range.

The guidemark detection capability is primarily a supplement intended to improve the robot localisation and may solve positional ambiguities. In the initial phase for the robot, the start posture may be determined from a guidemark. Where accurate localisation is needed – eg when approaching a dogging station – a guidemark may provide the needed accuracy.

4.8 Further improvements

Further cameras and possibly manoeuvrable cameras may further improve the ability to detect side roads and interesting objects like natural or artificial guidemarks.

The vision-based perception process could include is much more then the two sensor types presented in this chapter. Further segmentation of the camera

image may provide additional information valuable for the navigation process. A combination with 'structure from motion' or stereovision techniques the positioning of the seen objects may be improved, and thus qualitatively improve the perception.

Chapter 5

Obstacle avoidance

5.1 Introduction

Obstacles are from the definition something that hinders progress, and in this project obstacles are primarily maintained for the purpose of obstacle avoidance.

The obstacle avoidance behaviour decisions should obey to overall behaviour requirements – eg follow left side of road – while avoiding detected obstacles. The data source is a number of possible corridors based on laser scanner traversable segments wide enough to allow passage by the robot, a vision-based road outline, and a number of detected obstacles that should be avoided. The result of obstacle avoidance should be a path within the area covered by the sensors that avoids the detected obstacles and brings the robot closer to the desired destination.

5.2 Obstacle avoidance methods

A number of methods for path planning to avoid obstacles are described in the literature and is adequately summarized by Siegwart & Nourbakhsh (2004). The methods describe different methods to calculate a route from current position to an exit position, the exit position being either the destination position or a position that would bring the robot closer to the desired destination.

The obstacle avoidance methods are divided into four groups:

- Road map: identify a route in the identified traversable area to a target position. Two basic methods are described: the direct visibility graph method and a Veroni diagram method.

  The visibility graph method connects all vertices of the obstacles with all visible vertices of all other obstacles including the current position

and the exit position. The shortest graph from the current position and the exit position is then the optimal route. If the number of obstacles is sparse the method is efficient, but the computation time grows fast with the number of obstacles.

The Veroni diagram makes a path from current position to the exit position following lines where there is equal distance to the two closest obstacles. If there is a route to the destination the method will find it, but the distance is not optimal, and especially in sparse environments, the solution is far from optimal.

- Cell decomposition: divide the area that includes the current and the exit position into obstacle cells and traversable cells. Two basic methods are described: exact cell decomposition and approximated cell decomposition.

  The exact cell decomposition divides the area between obstacles into areas, each cell has three or four sides, each cell is either traversable or an obstacle, and thus each side of a traversable cell is shared with another traversable cell or an obstacle. A graph can then connect the cell with the current position to the cell with the exit position, and an optimal sequence of cells be found. The method is efficient in large sparse environments, but rarely used due to implementation complexity.

  The approximate cell decomposition method uses square cells, where a cell is declared nontraversable if parts of the cell is occupied by an obstacle. The traversable cells are then searched from current position in a breast first search until the destination cell is found. The cell size should be small, so that narrow passages are not lost due to the cell approximation, but the cell count should be kept low to reduce the memory and processing time is needed. The implementation is simple and the method is popular.

  The cell size of the approximate cell decomposition method may be selected to be variable. Large free areas are then represented by one large square cell, and if obstacles are found inside this area then the area is split into four smaller cells. This cell splitting can be continued until the desired accuracy is obtained. The number of required cells is in this way reduced at the expense of implementation complexity.

- Potential field: using mathematical functions describing a field where the destination is attractive and the obstacles are repulsive. The current position is then a particle in this field, and the steepest decent is then the optimal route to the destination. Concave obstacles or obstacle groups may introduce local minima and thus trap the robot.

- Bug navigation: aiming for the destination and dealing with one obstacle at a time as they come. These methods may get to the exit position, but the route may be far from optimal.

5.2.1 Design decisions

The selected method is based on the visible graph method. The method is modified to ignore obstacles too far away and to start the path search with the direct route. The search may end here if no obstacles are found, or may be extended when the direct route crosses or gets too near an obstacle. The path is further including robot dynamics during the search.

The reasoning is that the method is intuitive simple, based on a graph of visible segments from current position to the exit position. The environment used by the robot is – apart from the road sides – a sparse environment, and thus a method favouring sparse environments could be selected. The method is scalable and allows the exit position to be far away from current position without increased performance penalty – as long as the obstacle density remains low.

The force field method may experience trapping by local minima, and should be combined with another method to recover from those situations. This method is therefore not further investigated.

The cell decomposition method using fixed sized cells may be relative simple to implement, but a high number of cells is needed. A coverage area of 15×15 m with a 5 cm resolution would require $300 \times 300 = 90000$ grid cells of which most need recalculation for each iteration. The method is not easily scalable is the exit position is far away. An improved scalability could be obtained by a variable cell size, but at the cost of complexity. The found route is further not easily modified to account for robot dynamics.

The accurate cell decomposition method could be a tempting compromise between the approximate cell decomposition method and the visibility graph, as it easily provides prioritised alternative routes if the shortest route is undesirable due to robot dynamics requirements. The method is however not further investigated.

5.3 Obstacle detection

On the experimental platform both the vision sensor and the laser scanner are capable of obstacles detection.

The implemented feature detection for the vision sensor is limited to road outline sensing. The identification of a road limitation – a transition from road to non-road – could be used as a detected obstacle, the accuracy of which is

limited primarily by the projection errors. The road outline is projected to a virtual flat surface in the same plane as the robot base. The projection therefore includes errors when the robot tilt and roll are different from zero and when the road surface is not flat. The camera is looking forward with a horizontal view angle of 58°, and as the main view area extends from the end of the laser scanner range – 2.6 m in front of the robot on a flat road – to the horizon, the main projection error is expected to be in the x-direction (forward). A tilt error φ_e of the robot probagates to an x-distance error x_e as shown in (5.1)

$$x_e = \frac{h}{\sin(\sin^{-1}(\frac{h}{r}) + \varphi_e)} - r \ , \tag{5.1}$$

where the robot tilt φ_e is approximated as camera tilt and x_e is approximated as error in distance along the camera axis. This is justifiable as the distance from the camera r is much larger than the camera height $h = 0.86$ m.

For an obstacle detected at $r = 10$ m and a tilt error $\varphi_e = -1.3°$ (corresponding to the front wheel being lifted 1 cm) is thus approximately 3.7 m.

An error of this magnitude makes it difficult to track and correlate obstacles based on the vision road outline sensor. Improvements in the robot posture sensing to include tilt could improve the situation.

Obstacles are further assumed to be flat and the (possible) road behind obstacles is not detected as traversable, this further complicates tracking and usage of obstacles based on the road outline detection.

The laser scanner based road classifier divides the measurements into traversable, nontraversable and invalid data, and further group measurements classified nontraversable into spatially separated polygon-shaped (convex) obstacles as described in section 3.7.

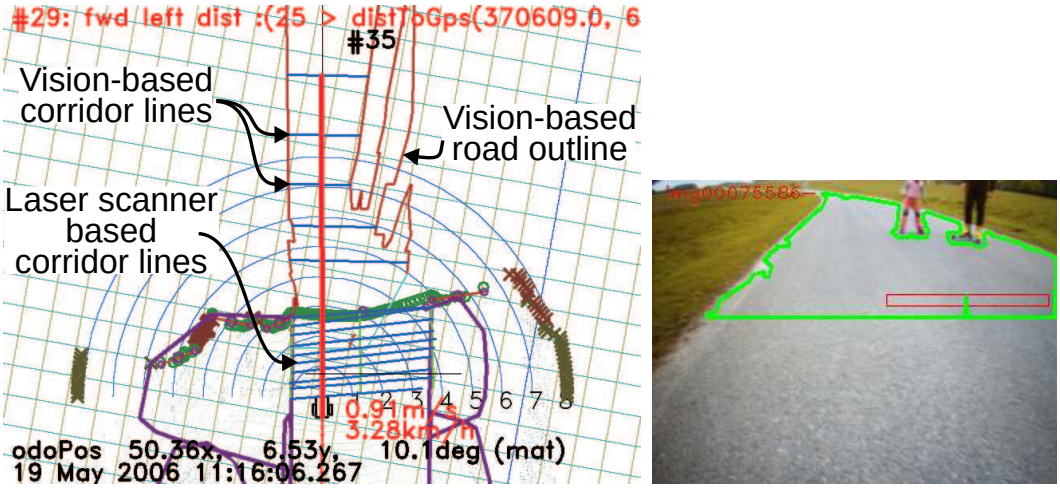
The projection error from the laser scanner measurements due to unknown robot tilt and roll is much less than for the vision sensor, as the laser scanner is a 3D sensor.

5.4 Integration of vision data

The obstacle avoidance solution needs a desired exit position and a set of obstacles. A combined corridor is found using both vision and laser scanner data. The exit position is found in the vision part of the corridor, so that any obstacle areas detected by the vision sensor are avoided. The visible graph method is used to find the optimal route to this exit position.

5.4.1 Extended road corridor

The laser scanner based road corridors may be extended into the vision sensor road outline. This may disqualify some of the corridors found by the laser



(a) (b)

Fig. 5.1: The corridor lines (blue in (a)) are based on laser scanner data up to the most recent laser scan, and from here extended into the vision-based road-outline polygon. The road-outline polygon is further shown in the camera image (b).

scanner, eg if a corridor was found in the cut grass on the roadside, then this is unlikely to correlate with the vision-based road outline.

The extension is calculated by creating a number of corridor line segments inside the vision polygon and correlates these with the laser scanner based corridor. The two sets of corridor lines are shown in Fig. 5.1.

The vision corridor consists of a number of traversable line segments S_r^v crossing the road-outline polygon at range r. The segments are generated as shown in (5.2)

$$S_r^v = \left\{ \begin{array}{c} S_{r,j}^v \\ j = [1..N/2] \\ m = 2j \end{array} \middle| \begin{array}{l} S_{r,j}^v = (\mathbf{x} + \mathbf{v}t) \\ \mathbf{x} = \mathbf{U}_{m+1} \\ \mathbf{U}_m = [r, y_m]^T \\ \mathbf{v} = [0, 1]^T \\ t = y_m - y_{m+1} \\ |y_m - y_{m+1}| > W_{\min} \end{array} \right\} , \tag{5.2}$$

where $[r, y_m]^T$ is the mth of N crossings of the road-outline polygon at range r from the robot. The crossing line at r is perpendicular to the current robot heading. $\mathbf{U}$ is the set of crossings at r, and these are ordered after decreasing y value. Each line segment $S_{r,j}^v$ is described in robot coordinates. The segment must be wider than $W_{\min} = W_R$ (robot width), except if one of the positions y_m or y_{m+1} is at the image border, in which case $W_{\min} = W_R/2$ to allow for the possibility that the rest is outside the image.

The segments are generated at the range intervals

$$r = \left\{ r_i \;\middle|\; \begin{array}{l} r_1 = r_{\text{seed}} + r_{\text{sep}} \\ r_i = r_{i-1} d_{\text{sep}} \\ r_i < r_{\max} \\ i \in [1, i_{\max}] \end{array} \right\}, \tag{5.3}$$

where r_{seed} is the distance from the seed area used for the road outline (typically about 2.6 m), and $r_{\text{sep}} = 2.0\,\text{m}$ is a fixed distance to the first vision-based segment, the segments are separated by a factor $d_{\text{sep}} = 1.2$ until a maximum range $r_{\max} = 14\,\text{m}$ is reached or there are no more crossings of the outline polygon.

If there are no polygon crossings at r_1, then no intervals are generated and the vision data are ignored.

If the destination position is closer than 14 m, then $r_{\max}$ is reduced accordingly. The value $r_{\max} = 14\,\text{m}$ gives the maximum reaction distance to obstacles, ie the distance where an obstacle avoidance manoeuvre may be initiated as the earliest. The distance is also the longest distance at which vision road outline can assist in junction resolutions.

The segment interval corresponds to a constant obstacle height D_o^v as shown in (5.4)

$$D_o^v = h_{\text{cam}} \frac{d_{\text{sep}} - 1}{d_{\text{sep}}} = 0.15\ \text{m}, \tag{5.4}$$

where h_{cam} is the camera height. This is more than used by the laser scanner obstacle detection, and should probably be reduced if the road outline detection was modified to test for obstacles along these line segments. Such – rather simple – modification to the road outline detector would improve the obstacle detection capabilities to about the same level as the laser scanner (but is not implemented).

The corridor consists of segments from (5.2) correlated as in (3.23) and continued into the laser scanner corridor (3.28). The full corridor B_j consists of line segments as in (5.5)

$$B_j = \{S_1, S_2, \ldots, S_n, S_{n+1}, \ldots, S_{n+m}\}\,;\ j \in [1, b], \tag{5.5}$$

where $S_1 = S^v_{r_1,j}$ from (5.2) to reduce the notation complexity, and n is the number of vision-based segments. Similarly $S_{n+1} = S_i^k$ from (3.28), and m is the number of laser scanner based segments in the corridor.

The forward most vision range line may be broken into more segments resulting in b available corridors. The segments can also be described as parameter lines:

$$\begin{aligned} S_i = \mathbf{s}_i(t) &= \{\mathbf{x} + t\mathbf{v} \mid t \in [0, l_i]\} \\ \mathbf{x} &= [x_{xs}, x_{ys}]^T \\ \mathbf{v} &= [v_{xs}, v_{ys}]^T\ , \end{aligned} \tag{5.6}$$

where l_i is the length of segment i.

5.5 Exit posture

The exit posture (exit position and desired exit heading) is the foremost usable aiming point inside the currently detected obstacle-free area. This posture is determined by the current navigation objectives and the navigation corridor. Two main types of navigation objectives are relevant here: following a road line and aiming for a specific position. A third type – direct to destination – is available too, especially for tight manoeuvres; the exit posture for this type is taken directly from the destination posture.

When the objectives are to follow a road line, the relevant road line $\mathbf{r}(t_r) = [\mathbf{x}_r + t_r\mathbf{v}_r]$ is extended to its crossing with the segments $\mathbf{s}_i(t_s)$ from the corridor B_j found in (5.5) and (5.6).

The crossing of these lines is expressed as a parameter value t_s on the segment $\mathbf{s}_i$ as in (5.7) and (5.8)

$$\mathbf{r}(t_r) = \mathbf{s}(t_s) \quad \Rightarrow \quad [\,\mathbf{v}_r,\ -\mathbf{v}_s\,] \begin{bmatrix} t_r \\ t_s \end{bmatrix} = [\,\mathbf{x}_s - \mathbf{x}_r\,]\,, \tag{5.7}$$

and solved for the parameter value t_s for the crossing of the segment yields:

$$t_s = \mathrm{cross}(\mathbf{r}, \mathbf{s}) = \frac{v_{xr}(x_{yr} - x_{ys}) + v_{yr}(x_{xs} - x_{xr})}{v_{xr}v_{ys} - v_{yr}v_{xs}}\,, \tag{5.8}$$

where $\mathbf{x}_r = [x_{xr}, x_{yr}]$ and $\mathbf{v}_r = [v_{xr}, v_{yr}]$.

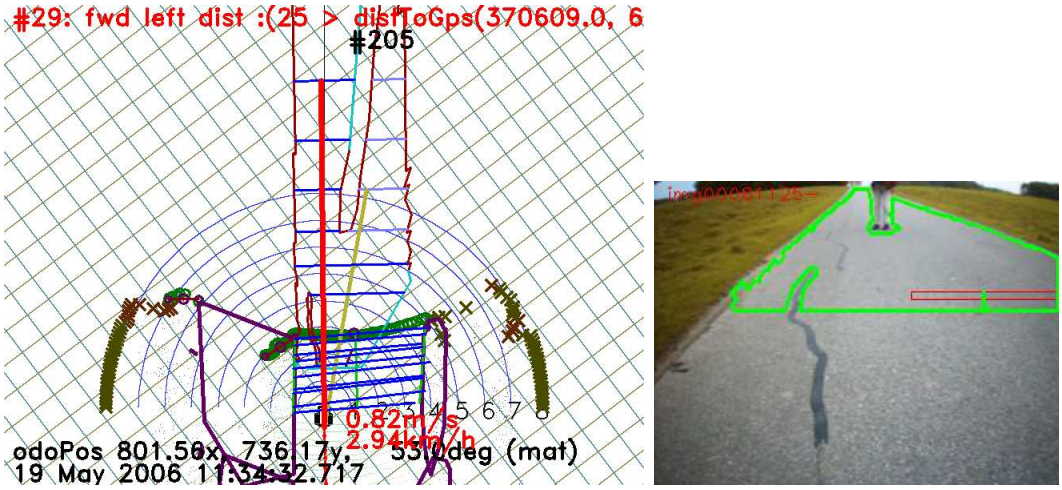
The exit point candidate $\mathbf{c}_i$ is then

$$\mathbf{c}_i = \mathbf{s}_i(t_{si} + d_r), \tag{5.9}$$

where t_{si} is the t_s value for segment i and d_r is the desired position on the road relative to the road edge.

If $\mathbf{c}_i$ is inside all segments in the vision parts of the corridor, then the exit position $\mathbf{p} = \mathbf{c}_1$. If one of the crossings is outside the vision segment or closer to its end points than a security distance $d_{\mathrm{sec}} = 0.3\,\mathrm{m}$, then the candidate exit position $\mathbf{c}_i$ is adjusted to be inside the segment before the next candidate position is tested - as shown in (5.10)

$$\mathbf{p} = \mathbf{c}_n \left| \begin{array}{l} \mathbf{c}_0 = \mathbf{c}_1 \\ \text{for } i \in [1, n]: \\ \mathbf{r}_{cc} = [\mathbf{x}_c + t_c(\mathbf{c}_{i-1} - \mathbf{x}_c)] \\ t_{si} = \mathrm{cross}(\mathbf{r}_{cc}, \mathbf{s}_i) \\ \mathbf{c}_i = \left\{ \begin{array}{ll} \mathbf{c}_{i-1} & \text{for} \left\{ \begin{array}{l} t_{si} > d_{\mathrm{sec}} \\ \wedge\, t_{si} < l - d_{\mathrm{sec}} \end{array} \right. \\ \mathbf{s}_i(d_{\mathrm{sec}}) & \text{for } t_{si} < d_{\mathrm{sec}} \\ \mathbf{s}_i(l_i - d_{\mathrm{sec}}) & \text{for } t_{si} > l_i - d_{\mathrm{sec}} \end{array} \right. \end{array} \right. , \tag{5.10}$$



(a) (b)

Fig. 5.2: Two available corridors are generated form this scene. The exit position in the right alternative (yellow in (a)) is moved to the segment just next to the obstacle. The left corridor follows the left edge of the road and is not sufficiently close to the obstacle to move the exit position. The corresponding camera image is shown in (b).

where $\mathbf{x}_c$ is the current robot position.

When the objective is to aim at a position rather than following a road line the exit position is found using (5.10) with the initial candidate position being $\mathbf{c}_1 = \mathbf{s}_1(\text{cross}(\mathbf{r}_{cd}, \mathbf{s}_i))$ where $\mathbf{r}_{cd}$ is the line from the current robot position to the destination.

The exit posture also includes a desired exit heading. When the objective is to follow a road line then the road direction is used as the exit heading. If the objective is a position then the exit heading is set to the direction from the exit position to the destination position.

5.5.1 Exit posture results

In Fig. 5.1(a) the path line (bold red) extends to the exit position, in this case the $\mathbf{c}_1$ position. The robot is following the left road edge at a distance $d_r = 0.85\,\text{m}$. Figure 5.2 shows two available corridors where the exit position to the right is adjusted to a position just to the right of the obstacle.

5.6 Obstacle maintenance

Obstacles found by the vision system are now avoided by adjusting the exit position as described in the previous section.

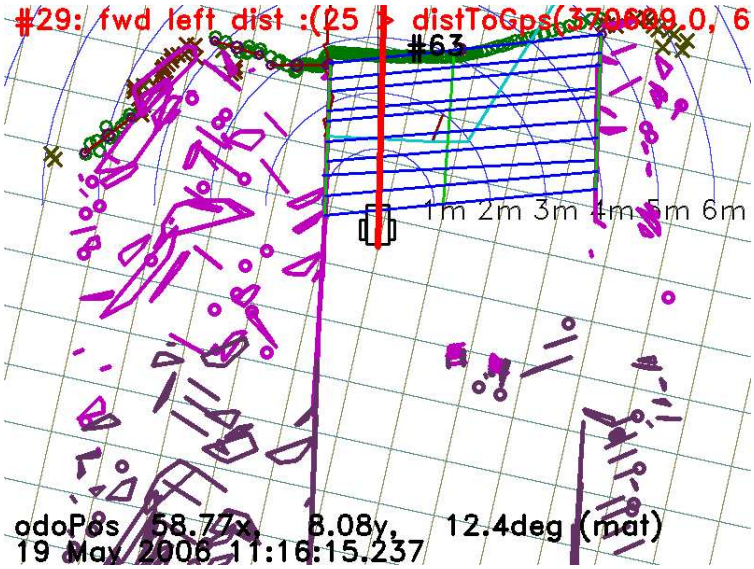


Fig. 5.3: Single scan obstacles detected while passing two pedestrians on an asphalt road (A little further ahead than the scene in Fig. 5.1). Single measurement obstacles are shown as circles. Two groups of obstacles are shown in slightly different colours.

The obstacles detected by the laser scanner are combined into polygons describing obstacles and other nontraversable areas – as described in section 3.7. In addition to these obstacles the left and right road edge lines – as described in section 3.6.3 – are added as obstacles if the line is detected with a sufficiently high quality $L_q > 0.9$.

The obstacles from especially the rough grass are often seen as isolated patches, and at every new scan different grass patches are detected as shown in Fig. 5.3. The high number of obstacles shown in the image could be reduced by combining near obstacles detected in different scans. This would reduce the number of obstacles that need to be handled and would combine the nontraversable rough grass areas into larger obstacle areas more in line with the intuitive perception of the area.

The obstacles need to be maintained until these are no longer relevant for obstacle avoidance. The accumulating errors from the robot posture estimation make the detected obstacle positions gradually more imprecise as a function of the distance driven since detection. As a consequence the obstacle positions should either be corrected or the obstacles ignored before the accumulating errors prevent progress.

The obstacles detected over a limited robot travelling distance $d_{\text{lim}} = 7\,\text{m}$ or a limited detection time $t_{\text{lim}} = 7\,\text{s}$ are grouped. Obstacles detected – within this group – are attempted correlated with new obstacles.

A new obstacle polygon D is tested for correlation with an established obstacle polygon E using a proximity function $\mathcal{F}(D, E)$ as shown in (5.11). This function evaluates the distance from the closest vertex in D to any edge in E. The distance is signed so that a positive value is evaluated if the nearest vertex in D is outside the polygon E. The polygons are all convex and vertices are ordered counter-clockwise

$$
\begin{aligned}
&\mathcal{F}(W,G) = \overset{j\in[1,m]}{\min}\left[\overset{i\in[1,n]}{\max}\ (A_i w_{j,x} + B_i w_{j,y} + C_i)\right] \\
&\text{where} \\
&W = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n\};\ \mathbf{w}_i = [w_{i,x}, w_{i,y}]^T \\
&G = \{\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_m\};\ \mathbf{g}_i = [g_{i,x}, g_{i,y}]^T \\
&L = \{\mathbf{l}_1, \mathbf{l}_2, \ldots, \mathbf{l}_m\} \\
&\mathbf{l}_i = [A_i, B_i, C_i] \left| \begin{array}{l} A_i x + B_i y + C_i = 0 \\ \mathbf{d} = [d_x, d_y]^T = \mathbf{g}_i - \mathbf{g}_{i-1};\ \mathbf{g}_0 = \mathbf{g}_m \\ A_i = -\frac{d_y}{|d|};\ B_i = \frac{d_x}{|d|};\ C_i = \frac{d_y * g_{i,x} - d_x * g_{i,y}}{|d|} \\ i \in [1, m] \end{array} \right. .
\end{aligned}
\tag{5.11}
$$

A correlation is said to exist if $d_{\text{limit}} > \min[\mathcal{F}(D, E), \mathcal{F}(E, D)]$. The rough road edges are predominantly flat, and in many cases obstacles representing the same rough area do not overlap. The distance between obstacle detection in such areas depends on the distance between scans. If the area was a homogeneous area of single (nontraversable) straws the distance between obstacles in successive scans would be equal to the robot movement (of this position) between the scans. To allow for correlation between scans the parameter d_{limit} could be set to the maximum movement, but as the movement is known a different approach is taken. A point is added to the new polygon, at a position where this obstacle would have been detected in the last scan had the obstacle been a homogeneous nontraversable rough area. The back-most point in the candidate polygon (in robot local coordinates) is therefore added at the position it would have had if detected in the previous scan (the extra point is removed after the correlation process).

In addition to this extra point the limit $d_{\text{limit}} = 0.3\,\text{m}$ is found experimentally to get a reasonable reduction in obstacle count while keeping other types of distinct obstacle separate.

The resulting set of obstacles after correlation is shown in Fig. 5.4 for the scene in Fig. 5.3.

Similar results are obtained on a gravelled road as shown in Fig. 5.5. The scene is a gravelled road with rough edges just before it opens for a side road to the left. The road edge lines are mostly of poor quality. The obstacles are again combined to almost solid areas on both roadsides.

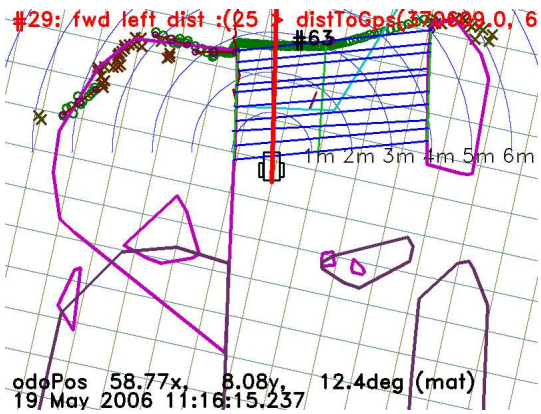
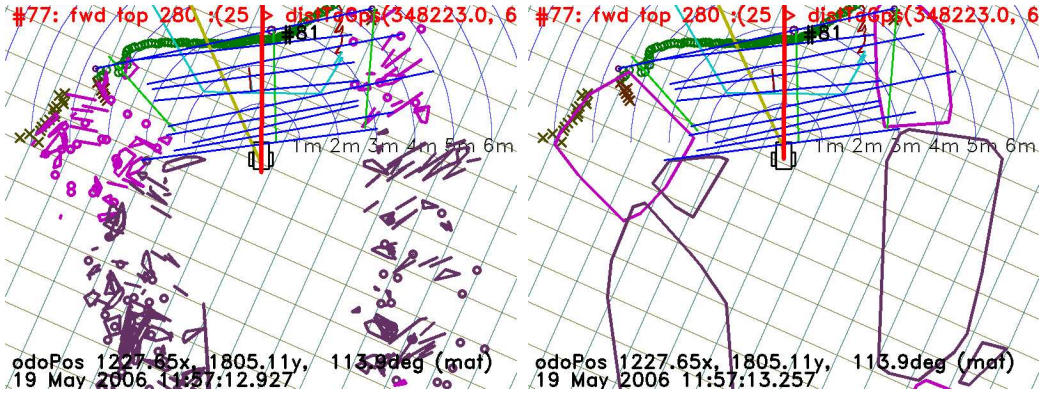


Fig. 5.4: The correlated and merged obstacles shown represent the same obstacles as shown in Fig. 5.3. Two obstacle groups are shown (in slightly different colours).



(a) (b)

Fig. 5.5: The scene is a gravelled road with rough road edges. The raw obstacles in (a) are combined into nontraversable obstacle areas in (b).

The decision to use convex polygons to describe obstacles was to ease the computation, the drawback is that the obstacle may be concave by nature and the area in this concavity may be needed for navigation. An example could be that the obstacle extends alongside the robot and an obstacle at the roadside in front of the robot is detected, this obstacle is merged with the one at the side of the robot to one convex obstacle, now including the robot position. This situation would trigger unnecessary abrupt robot manoeuvres to get the robot out of the obstacle.

There is presently no provision for removal of obstacles nor to reduce the obstacle size, this would be needed if the obstacles were to be retained for a revisit of the area. Moving obstacles – like humans – are not tracked and may thus produce a series of obstacles. Such obstacles could block the robot

progress, but will eventually be too old to be considered.

5.7 Obstacle avoidance routes

The exit posture of the current manoeuvre is determined by the route objectives as described in section 5.5. The obstacle avoidance route is then planned from current position to the desired exit posture.

The obstacle avoidance route is determined as an iterative calculation, starting with the expected position a fixed time ahead of current position and ending at the desired exit position. The calculated route may be impossible due to detected obstacles; when this is the case the route is recalculated avoiding one obstacle (the most critical) at a time until a route is found that avoids all detected obstacles.

Obstacles are initially avoided at the expected shortest path – left or right – around the obstacle (based on the obstacle centre of gravity). The most offending part of the obstacle is found and a new mid-posture for the route is calculated, ie a posture that would be safe for the robot to pass, as shown in Fig. 5.6.

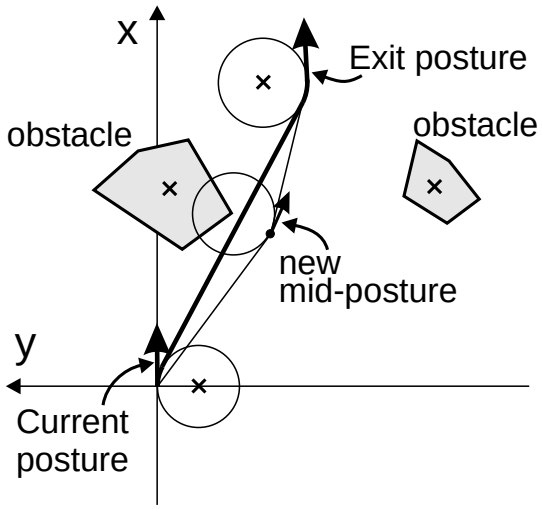


Fig. 5.6: Obstacle avoidance starts by planning a route to the exit that ignores obstacles and then avoiding obstacles on the planned route in a safe distance, introducing new mid-postures on the route.

If there is another obstacle close to the most offending position, and the distance between the obstacles do not allow passage, then this obstacle is avoided too.

If the distance between the obstacles just allows passage, then the new mid-posture is planned just between the obstacles at a heading that should give the

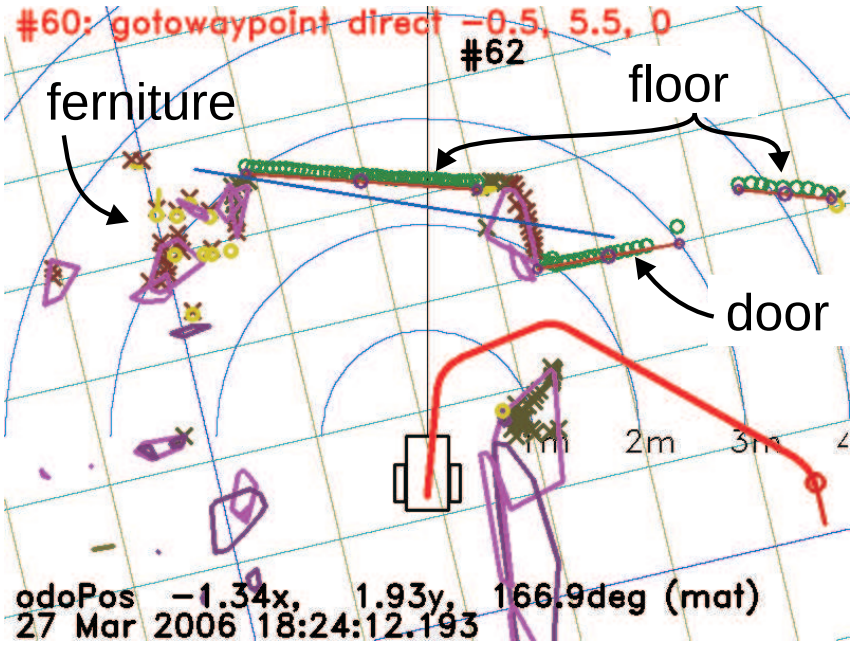


Fig. 5.7: Obstacle avoidance path planning in an indoor scene. The path planner has found a path out of an open doorway avoiding the known obstacles at the time. The door is erroneously classified as traversable.

safest passage, ie the new mid-posture is the centre of a line connecting the obstacles, and the heading is in the direction of the mid-normal.

The less obvious side of the obstacle is evaluated too, as this may be the best overall guess.

The new mid-posture splits the path from the robot to the desired exit posture in two. Each of these are then recursively tested for free passage and recalculated as needed.

An example of path planning is shown in Fig. 5.7. The robot has planned a route through a door opening in an indoor scene. The outdoor obstacle avoidance routes are usually less complicated. The setup is not intended for indoor use, but the indoor scene is handled reasonably well after introduction of two minor exceptions: obstacles are not generated for ditches and the obstacle correlation distance d_{limit} is reduced to 0.03 m to reduce the likelihood of two walls in a corner are correlated into one convex obstacle, as this potentially could limit the free area for the robot significantly.

A number of exceptions are introduced to maintain a smooth path and a fast calculation. The number of alternative paths (the other way around obstacles) is limited (to 6). The number of iterations finding new mid-postures is limited too (to 6). This means that a path may be rejected if the calculation is too complicated (but a path may exist). When a new mid-posture is generated close

to an existing one, the old mid-posture is probably not optimal for passage and is replaced by the new.

5.8 Posture to posture manoeuvre

Obstacle avoidance manoeuvres are planned using a combination of straight and curved paths. The manoeuvres from one posture to another will always be planned using a combination of an arc, a straight part and a final arc. The arcs can either form a left or a right turn. The arc radius is – when possible – planned to limit the centripetal acceleration.

Both the centripetal and the lateral acceleration are limited in order to get smooth manoeuvring.

The basic calculation of a posture to posture manoeuvre can be evaluated as one of two cases: both arcs are in the same direction or the arcs are in different directions. That is, right-straight-right or right-straight-left. The same manoeuvres starting with a left turn are calculated as a mirror image of the corresponding manoeuvre starting with a right turn. The manoeuvres are calculated in a reference coordinate system aligned with the start posture. The end posture is then (x_2, y_2, θ_2) as shown in Fig. 5.8 and Fig. 5.9.

The turn radius r is selected not to exceed a desired centripetal acceleration a_c, for normal manoeuvring as shown in (5.12) for the actual robot velocity v

$$r = \frac{v^2}{a_c}. \tag{5.12}$$

The arcs ϕ_1, ϕ_2 and the line d are found as described below.

5.8.1 Right, straight and then right

The manoeuvre consists of an arc ϕ_1 with a turn radius of r_1, a straight part d and a second turn (to the right) of ϕ_2 at a turn radius r_2, as shown in Fig. 5.8(a).

The solution is found by shifting the line d a distance r_2 (into d' in Fig. 5.8), so that it starts at the circle centre C_2 and ends at the tangent point to a circle with centre at C_1 and radius $r_1 - r_2$. The right angled triangle with this line (d') and the radius $r_1 - r_2$ as the two smaller sides and the line $\overrightarrow{C_2C_1}$ between the circle centres C_1 and C_2 as the longer side is solved, and from that finding

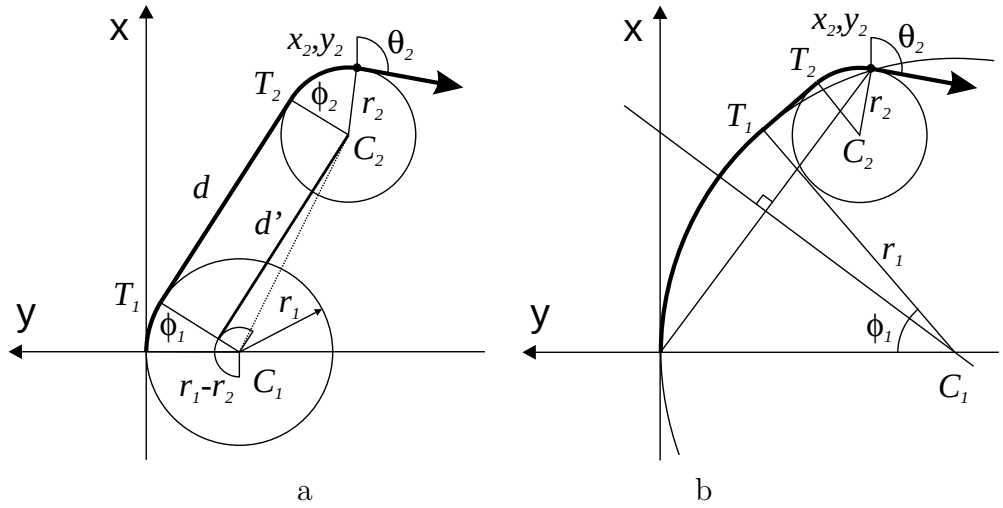


Fig. 5.8: Calculation of right-straight-right manoeuvre (a) and the same manoeuvre with the largest possible entry turn radius (b). The manoeuvres start at $(0, 0)$ heading 0°. The exit point is at (x_2, y_2) heading θ_2. The first arc stops at T_1, then straight until T_2 where the second arc starts.

the arc sizes ϕ_1 and ϕ_2 as shown in (5.13)

$$\begin{aligned} \angle(\overrightarrow{C_1T_1} &= \angle(\overrightarrow{C_2C_1}) - \sin^{-1}\frac{r_1 - r_2}{|\overrightarrow{C_2C_1}|} - \frac{\pi}{2} \\ \phi_1 &= \frac{\pi}{2} - \angle(\overrightarrow{C_1T_1}) \\ \angle(\overrightarrow{C_2T_2} &= \angle(\overrightarrow{C_1T_1} \\ \phi_2 &= \angle(\overrightarrow{C_2T_2}) - (\theta_2 + \frac{\pi}{2}). \end{aligned} \tag{5.13}$$

The limitations for this manoeuvre are twofold: The first limitation is a construction limitation: the turn radius must be so small that the exit position is not within the first circle and vice versa. The second limitation is more practical, the exit angle θ_2 should be smaller than the tangent angle $\angle(\overrightarrow{C_2T_2})$, otherwise the second arc could be up to a full circle. In such cases another manoeuvre should be selected. At the same time the tangent angle $\angle(\overrightarrow{C_1T_1})$ should be smaller than $\frac{\pi}{2}$ to avoid too large arcs at the start of the manoeuvre.

The last limitation corresponds to the situation where the second circle crosses the x-axis.

5.8.2 Right, straight and then left

The calculation of the manoeuvre starting with a right turn followed by a straight part ending in a left turn is very similar to the right-straight-right

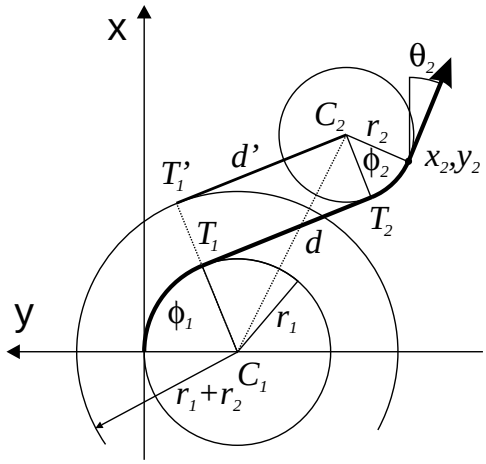


Fig. 5.9: Calculation of right-straight-left manoeuvre. The manoeuvre starts at $(0, 0)$ at heading 0°, and ends at position (x, y) heading θ_2. The maximum turn radius is when the two circles with radii r_1 and r_2 meet.

manoeuvre, except that the support line d' is now shifted away from the C_1 centre.

The revised calculations are shown in (5.14)

$$\begin{aligned}
\angle(\overrightarrow{C_1T_1'} &= \angle(\overrightarrow{C_2C_1}) - \sin^{-1}\frac{r_1 + r_2}{|\overrightarrow{C_2C_1}|} - \frac{\pi}{2} \\
\phi_1 &= \frac{\pi}{2} - \angle(\overrightarrow{C_1T_1'}) \\
\angle(\overrightarrow{C_2T_2} &= \angle(\overrightarrow{C_1T_1'} - \pi \\
\phi_2 &= \theta_2 - \frac{\pi}{2} - (\angle(\overrightarrow{C_2T_2})).
\end{aligned} \tag{5.14}$$

The manoeuvre turning radius r_1 and r_2 must not be so large that the two circles overlap.

The calculated posture to posture manoeuvre can now be used for calculation of proximity to obstacles as described in the previous section.

5.9 Route selection

The obstacle avoidance planner may provide a number of possible obstacle avoidance routes, as there may be more alternative corridors and thus more than one possible exit posture. Each exit posture may further result in a number of different obstacle avoidance routes. The available routes are therefore

rated using a quality R_w calculated as

$$R_w = \frac{\sum_{i=1}^{N} w_i q_i}{\sum_{i=1}^{N} w_i} \,, \tag{5.15}$$

where $q_i \in Q_r$ and $Q_r = \{q_\sigma, q_{\varphi 1}, q_{1\text{–}5}, q_e, q_w, q_d, q_m\}$ is the set of road qualifiers and w_i is the corresponding weight factor for q_i. Each value is described below.

Corridor roughness is used as a qualifier q_σ to prefer smoother corridors. The value is based on the average of the minimum of the roughness values R_n found in each of the laser scanner based traversable segments B_j in the corridor

$$q_\sigma = \left(1 + \frac{s_\sigma}{N} \sum_{j=1}^{N} (\min(R_n \in B_j))\right)^{-1} \;;\; w_{\sigma 1} = 3 \,, \tag{5.16}$$

where $s_\sigma = 20$ to scale the roughness for a reasonable quality interval.

The start manoeuvre is preferred as straight as possible; the angle to the first manoeuvre mid-posture (Φ_1) is used as a source for the start manoeuvre quality

$$q_{\varphi 1} = (1 + \Phi_1 - \varphi_d)^{-1};\; w_{\varphi 1} = 1 \,, \tag{5.17}$$

where φ_d is an offset used to prefer some directions. If the robot is to follow left or right side of the road the offset is set to $\varphi_d = +10°$ or $\varphi_d = -10°$, correspondingly. If the exit posture is behind the robot the offset is set to prefer $\varphi_d + 70°$ or $\varphi_d - 70°$ in the most appropriate direction.

The historic heading is calculated from the general direction φ_h travelled 1–5 m back from the current position. The φ_h is calculated as the direction of the best fit line for the robot positions recorded for this interval and the qualifier as

$$q_{1\text{–}5} = (1 + \Phi_e - \varphi_h)^{-1};\; w_{1\text{–}5} = 2 \,, \tag{5.18}$$

where Φ_e is the direction from the robot to the exit posture.

The road edge quality is used when the robot is following a road edge and is based on the relevant road edge quality L_Q (from section 3.6.3) as

$$q_e = \begin{cases} L_{Q,\text{left}} & \text{when following left road line} \\ L_{Q,\text{right}} & \text{when following right road line} \\ L_{Q,\text{top}} & \text{when following road centre line} \\ 1 & \textit{otherwise} \end{cases} \;;\; w_\text{e} = 1.5 \,. \tag{5.19}$$

The corridor width is used as a qualifier based on the road width R_w from the laser scanner based corridor. It is used to prefer wider corridors (up to 5 m) and calculated as

$$q_w = (6 - \min(5, R_w))^{-1};\ w_\mathrm{w} = 0.3\ . \tag{5.20}$$

The path distance is used as a qualifier to prefer routes with a distance equal to the direct distance to the expected target position d_d using

$$q_d = \begin{cases} (1 + d_m - d_d)^{-1} & \text{for} \quad d_m \geqq d_d \\ (1 + d_m - d_d)^{-2} & \text{for} \quad d_m < d_d \end{cases};\ w_d = 1\ , \tag{5.21}$$

where d_m is the path distance. The desired distance d_d is the distance to the exit posture, except if no vision solution is available and the manoeuvre is not for a direct destination. In these cases the exit posture is on a laser scanner traversable segment, and the preferred distance is the distance at which the laser scanner is expected to see the road (typically 2.6 m). A shorter distance is thus possible, but this is disfavoured as it – as an example – may be the lower part of a newly detected wall.

The manoeuvre deviation σ_m from the straight line to the destination is used to calculate the manoeuvre qualifier

$$q_m = \left(1 + \sqrt{\frac{1}{K}\sum_{i=1}^{K} M_i^2}\right)^{-1};\ w_m = 1\ , \tag{5.22}$$

where M_i is the distance from the mid-posture i to the straight line from the robot to the exit posture. This qualifier favours straight path solutions.

The route having the highest quality is selected. In Fig. 5.10 four routes are generated to three exit postures. The short route (in red) is selected.

5.10 Drive commands

The drive scheduler takes a sequence of drive commands and transforms these into wheel speed commands in real time. The best of the paths created by the obstacle avoidance planner therefore needs to be translated into the set of commands used by the drive scheduler.

The first metre of the planned route is transferred to the drive scheduler (or less if the route is shorter). At a maximum speed of 1.5 ms^{-1} and a path calculation rate of 2 Hz the robot will travel 0.75 m at maximum before a new route is available.

Two types of drive commands are used to control the robot:

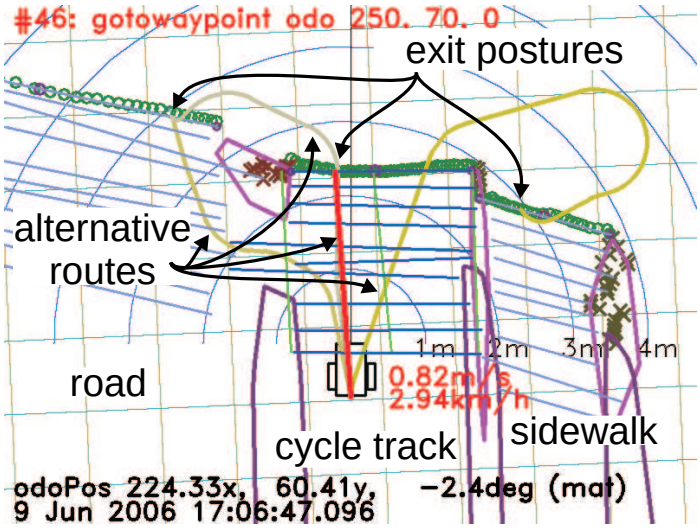


Fig. 5.10: The robot is driving on a cycle path with no vision support. Each of the road types (road, cycle path and sidewalk) generates an exit position relative to the objectives (a waypoint some 25 m (almost) straight ahead). Four routes are generated towards these points and the shortest (in red) is selected as the best.

Turn with a given radius is used directly as planned if the turn radius is less than 3 m (`turnr r a`, where `r` is the radius and `a` is the angle to turn).

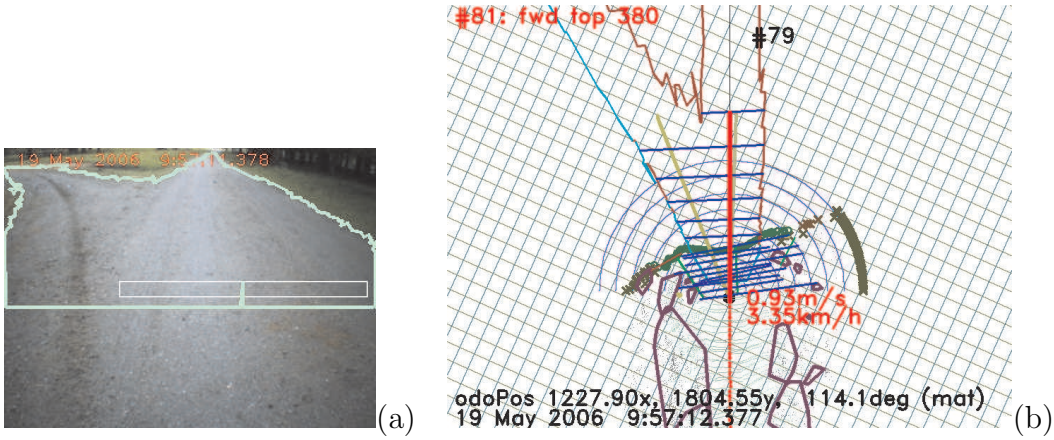
Following a target line, where the robot aims at following a provided line (`drive x y th`, where `x` and `y` is a position on the line and `th` is the direction of the line).

Turns with a turn radius larger than 3 m is divided into a sequence of line follow commands. Each command may include options for speed and acceleration. A stop criterion that – when fulfilled – makes the drive scheduler continue to the next command are used to limit the distance for each drive command.

The drive sequencer can execute a sequence of these (and other) commands, and supports that they are replaced by a new sequence when needed.

The obstacle avoidance route calculation takes a variable amount of time (but less than 0.3 s), and the robot posture used in the route calculation are therefore predicted 0.3 s along the last ordered path.

Before the path is sent to the drive scheduler the first part of the route (up to the first mid-posture) is recalculated using the most recent robot posture.



(a) (b)

Fig. 5.11: A fork is seen by the vision sensor (a) and included in the road outline shown on the feature map (b). The navigation process provides two main alternatives and selects the fork to the right.

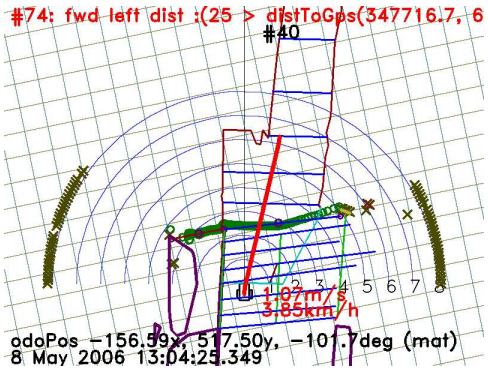


Fig. 5.12: A pedestrian blocks the vision-based road outline and an exit-posture next to the pedestrian triggers an early obstacle avoidance manoeuvre.

5.11 Summary

In junctions and transitions from open areas to a narrower road the vision will often be able to see the exit road or the road alternatives. This can assist the navigation system in planning a route towards the right road at an early stage. In Fig. 5.11 the vision-based road outline clearly show the road alternatives.

When the road outline is limited by obstacles as in the situation shown in Fig. 5.12, the route possibilities will be limited correspondingly, and the result is an obstacle avoidance route initiated at an early stage. The pedestrian can follow the robot intentions as soon as the obstacle avoidance route is initiated, and thus limits potential conflict situations.

The calculation of the obstacle avoidance route turned out to be more demanding than the simplicity of the approach suggested. The number of alter-

natives and exceptions make the implementation more difficult to test, and – unfortunately – the last error is not found in that piece of code.

The obstacle management ensures a reasonably low number of obstacles, and the history of obstacles aligns nicely with the road.

In the majority of the tested situations the obstacle generation and the obstacle avoidance route planning behave sensibly and produce the optimal result from the available data.

Chapter 6

Mission planning and execution

6.1 Introduction

This chapter deals with the mission preparation and navigation scheduling. In the previous chapters the sensors are described extracting obstacles and traversable areas, and how these obstacles and areas are used to generate a route that avoids the obstacles in search of a higher level goal. All that is missing is a scheduler that can break a long navigation mission into a sequence of short term navigation goals needed by the obstacle avoidance function.

A language is needed both when assigning a mission to a robot and when this mission is decomposed into a navigation sequence.

A language to implement missions using a language with a high abstraction level is described in Simmons & Apfelbaum (1998). This language supports job sequencing supports as well as parallel jobs. The parallel jobs are used for example when driving and speaking simultaneously, or for event monitoring. This and a number of other languages like Brooks (1990) has been defined over the years, but these are primarily used to implement a desired complete behaviour of the robot, ie the mission is compiled and loaded to the robot in executable form.

A language for the overall task allocation is described in Mizoguchi et al. (1999) where multiple robot agents in an office environment exchange and negotiate missions or tasks – the described case is handling and distribution of printouts from the office printer.

The language described in this chapter is somewhere in between, it is intended to control a navigation task for the current part of a mission. The current mission part may then be replaced by a new script, where the new script may be modified based on new planning information.

This chapter first discusses the situations that need to be handled and then continues describing the selected solution.

The selected solution is a navigation script language that can handle the

situations foreseen in the test area utilising drive system and the information provided by the sensors.

6.2 Mission

A mission is a task, a job, something you can send someone to do. It sounds simple, but there may be a number of subtasks that is implied by the mission. If the mission is to fetch some coffee, then the subtasks could be 'find coffee machine', 'find jug', 'pour coffee', 'return coffee', but it could also imply 'make coffee', if the machine is empty or 'buy more coffee beans' if need be.

Such tasks could be handled sequentially with a number if 'if' statements like 'if coffee machine is empty: make coffee' and 'if out of coffee beans: go get some'. But something may occur during execution of some subtasks, the route may be blocked, the battery condition may require a recharge or another mission may be assigned with higher priority.

The mission controller is assumed to be able to handle the overall division of the mission into tasks and to handle events that may change the mission priority. The tasks may be for different parts of the robot where for example navigation (motion) may be handled as one task, and separate from a manipulation task or a voice and voice recognition task – if such systems were available.

The navigation part of a mission – getting from A to B – may in the same way be divided into subtasks and be able to handle isolated event types related to navigation. Such events could be: the wheels is getting stuck, the robot is trapped in a dead end situation or a change in weather condition affecting the navigation abilities. Some of these may be handled by the navigation function and some may be reported to the mission level for possible re-evaluation of the mission.

This chapter deals with the navigation part of a mission and the preparation of navigation missions.

6.3 Navigation scheduler

The implemented navigation scheduler takes a navigation script with a combination of drive commands and other commands. The drive commands are mostly fed to the obstacle avoidance function, but may – if needed – be fed directly to the drive scheduler if the obstacle avoidance function needs to be bypassed. The other commands may be a setting change, a calculation, a status checking or a conditional jump in the script.

The navigation scheduler maintains a set of global parameters that all parts of the behaviour server may use (and maintain). These parameters include

desired speed, current position, drive mode, road width, laser tilt-angle etc.

A small script that follows a road until the road width is above 6.5 m and then stops could look like

```
speed = 1.2
fwd left 200 :((roadWidth > 6.5) and (roadQual > 0.85))
speed = 0.3
fwd direct 1
turn 180
smrcl idle
```

The script sets the maximum speed to $1.2\,\mathrm{ms}^{-1}$ and then moves the robot forward following the left side of a road for maximum 200 m, and shorter if the two explicit stop conditions are fulfilled. After the forward command is finished the speed is reduced to $0.3\,\mathrm{ms}^{-1}$ while driving one metre. Finally the robot turns 180° and stops.

6.3.1 Drive commands and stop conditions

The drive commands use three coordinate systems

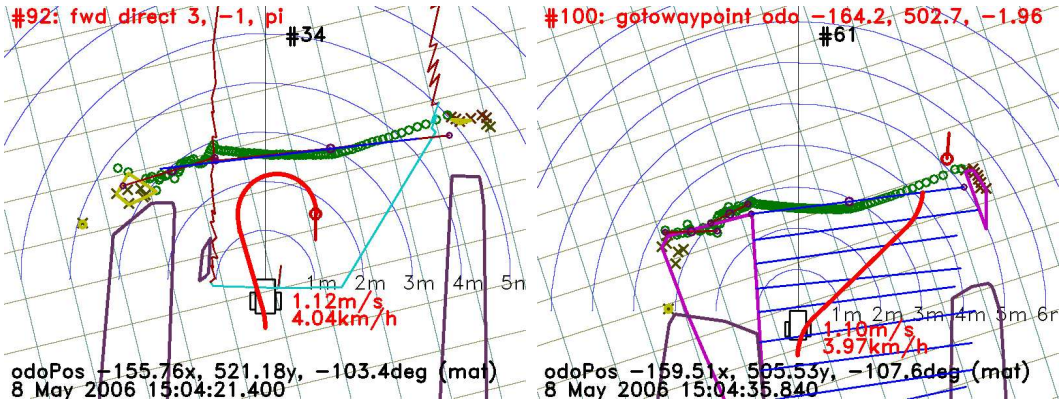
Relative coordinates $\mathbf{r} = [x_r, y_r]$ or posture $[x_r, y_r, \theta_r]$ are centred at the current robot position with x_r, y_r being forward and right, respectively. The direction θ_r is relative to robot heading with positive counter-clockwise.

Odometry coordinate $\mathbf{O} = [x_o, y_o]$ or posture $[x_o, y_o, \theta_o]$ with reference to the position and orientation where the robot started (was turned on), and subsequently as maintained by the drive server. The coordinate system will include all the position and heading (odometry) errors accumulated since the start.

GPS coordinates in UTM coordinates $\mathbf{U} = [E, N]$ (Easting, Northing) are available as position only, when a GPS heading θ_u is needed this heading can be obtained using a reference position some distance d back from current position.

A drive command can be specified in either relative or odometry coordinates towards a desired position or posture, as

`frd direct` x_r`,` y_r`,` θ_r
`frd odo` x_r`,` y_r`,` θ_r
`gotowaypoint direct` x_o`,` y_o`,` θ_o
`gotowaypoint odo` x_o`,` y_o`,` θ_o,



(a) (b)

Fig. 6.1: The forward relative command `fwd direct` plans a route that avoids obstacles but ignores the generated corridor (a). The `gotowaypoint odo` uses odometry coordinates and is in this case not `direct` and therefore follows the best available corridor (b). The red circle with a heading vector is the destination posture.

where `fwd` uses relative coordinates and `gotowaypoint` uses odometry coordinates. The `direct` qualifier is used primarily for close navigation, where the current corridor is ignored and the target posture is used directly as exit posture. An example is shown in Fig. 6.1(a). The vision support is not used with the `direct` qualifier. The `odo` qualifier forces the use of a corridor and finds an exit posture within the best suited corridor rather than aiming for the point directly (as in Fig. 6.1(b)), this is primarily used when the target position is some distance away and it is thus relevant to use the vision system to assist path planning.

The main command for topological navigation is to drive relative to one of the road lines. Three commands are available for this purpose

`frd left` d
`frd right` d
`frd top` d,

where d is the maximum distance to follow the road edge. The distance from the road edge is determined by a system variable controllable by the script.

The `top` qualifier makes the robot follow the centre of the road; the top is the detected highest point of the road. An example is shown in Fig. 6.2.

A number of drive commands are available that bypasses the obstacle avoidance path planning, as

`idle` t

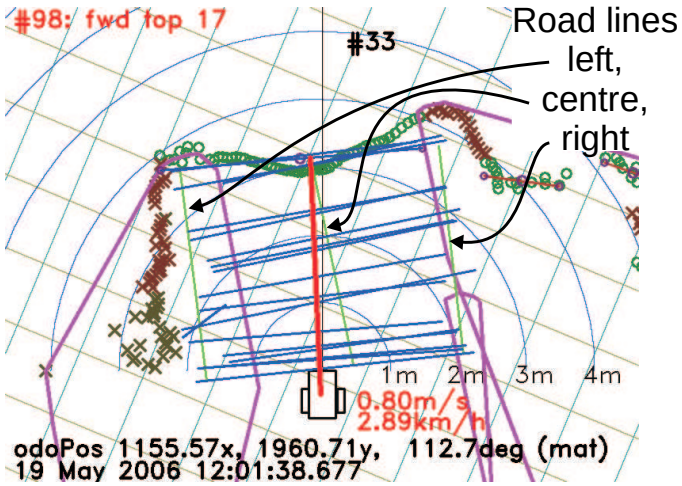


Fig. 6.2: The `fwd top` aims for the centre of the road. The centre is defined as the highest point of the road. The estimated road lines are shown as green lines, of which the central is used by this command.

```
smrcl 'turn ' φ
smrcl 'fwd ' 1.2 '@v-0.5'
```

The first command just idle the robot (and the navigation script) for t seconds. The `smrcl` commands are sent directly to the drive scheduler and follow the command syntax for this server. The first is a turn on the spot, the second is a reverse drive for 1.2 m at 0.5 ms^{-1}.

6.3.2 Stop conditions

Each of the above drive commands may additionally have an explicit stop condition to finish the command. A few examples are:

```
idle 6.0 :  (v_gmkget)
fwd left dist :(25 > distToGps(347716.7, 6183490.1))
gotowaypoint direct fwdx, fwdy, fwdh :(endPoseDist < 0.5),
```

where the first `idle` command waits for at most 6 s or shorter if the `v_gmkget` gets `true` earlier.

The `fwd left` command may continue for `dist` metres or finish when the distance to a given GPS position (in UTM coordinates) gets below 25 m. This illustrates the use of GPS, as the command will finish at the right spot within the accuracy limitation of the GPS (5 m in areas with good coverage), whereas the distance `dist` driven may be influenced by the number of obstacle avoidance manoeuvres needed.

The `gotowaypoint direct` command will drive for the given position, but as the robot may not move exactly as ordered and may pass the desired point at a distance, which further could trigger some undesired tight manoeuvres in an attempt to correct the (small) error. To avoid this situation it is often better to stop slightly – in this case 0.5 m – before the destination is reached.

The expression to the right of the ':' is evaluated as either `true` or `false` and will finish the command when evaluated to `true`. A number of variables and support functions are available when creating stop conditions.

The stop conditions (implicit or explicit) are evaluated at a 5 Hz rate.

6.3.3 Sensor control

The sensors – and here especially the vision sensor – may be controlled inside the script, using a server token in front of the command as

```
vision 'gmkget gmkref=true extra=false device=' dev
vision 'camget device=' dev ' pos rot focalLength',
```

where the command for the vision server is the remainder of the line after the `vision` keyword. The characters in apostrophes are sent directly to the vision server, whereas expressions outside the apostrophes are evaluated and replaced by a value.

The first command `gmkget` requests a guidemark analyses from the camera device `dev`. The second command `camget` requests the position (`pos`), rotation (`rot`) and focal length (`focalLength`) of the same camera device.

The commands do not wait for a reply – some image analysis may take more seconds – but when a reply is available a variable that includes the command name will be set eg `v_gmkget = true` when a `gmkget` reply is received. The returned values are in the same way available as variables; the reply to the `camget` command may thus generate and assign the following values:

```
v_camget = 1.0
v_camget_posX = 0.44
v_camget_posY = 0.0
v_camget_posZ = 0.86
v_camget_rotOmega = 0.0
v_camget_rotPhi = 0.392
v_camget_posKappa = 0.0
v_camget_focalLength = 297.5 ,
```

where `v_camget = 1.0` indicates that the reply is received; the value `1.0` is equivalent with `true`. All variable values are floating point numbers.

6.3.4 Control decisions

A control sequence alternative may be needed if the values obtained on the route may influence the actions needed. The following example uses the vision server to analyse for a guidemark, and dependent on the received code takes a different approach

```
try_gmk_again:
v_gmkget = false
t = time
dev = 0
vision 'gmkget gmkref=true extra=false device=' dev
//# wait for reply
idle 6.0 :(v_gmkget)
if (gmkNewCnt(t) < 1) goto got_no_gmk
if (gmkNewID(t, 13)) goto got_gmk13
if (gmkNewID(t, 11)) goto got_gmk11
got_no_gmk:
n = n + 1
if (n > 6) goto the_end
goto try_gmk_again
//
got_gmk11:
//# got gmk by right door -- enter
calcMapToPose(-2.0, 1.0, -3.14, gmkx, gmky, gmkk)
fwd direct calcx calcy calch :(endposedist < 0.3)
goto try_gmk_again
//
got_gmk13:
 ...
the_end:
print "Terminated"
```

The first lines request a guidemark analysis from the vision server, and then a bit down the command `if (gmkNewID(t, 11)) goto got_gmk11` an if statement that fetches the guidemark 11 from the guidemark database, if this is newer than the time t. If the guidemark is available then the next line to execute is after the label `got_gmk11:` found further down. There the guidemark position is used to calculate a position relative to the guidemark – in this case used to enter a door. After the next drive command a `goto try_gmk_again` sends the execution back to the beginning (to look for guidemark 13, that will do something else and terminate the script).

6.3.5 Watch functions

There may be a need to evaluate a condition that requires some calculation before a stop condition can be evaluated. One such condition could be to monitor the progress of the robot, and if it persistently gets further and further away from the target (eg while following a road edge) it may have taken a wrong turn and thus it could be time for a recovery action, or to inform the mission controller.

To enable such situation monitoring a watch function is implemented. The following example demonstrates the watch function to monitor for open doors while driving down a corridor. The intension is to make a pause at every open door.

```
speed = 1
door = true
doorNow = false
// define watch function
function look_for_door(direc)
  ld = laserRange(direc)
  doorNow = (ld > 2.5)
  if (not door and doorNow and (distSoFar > 1))
           skipcall doWait(5)
  if (door != doorNow) print "laser dist " ld
  door = doorNow
return door
//
function doWait(waitTime)
  dd = dd + distSoFar
  idle waitTime
return
// start watch for doors to the right
watch doors look_for_door(-90)
//
dist = 25
dd = 0
fwd left dist :((dd + distSoFar) > dist)
unwatch doors
print "Terminated"
```

A function is defined by the lines from a `function` to a `return`. The first function `look_for_door(direc)` tests a laser scanner range in the direction of the doors ($-90°$) and determines that there is an open door (or rather a lack of wall) if the measured distance is above 2.5 m. If a (new) door is detected

then the `if (...) skipcall doWait(5)` executes a `skipcall`. The `skipcall` terminates the current drive command temporarily and executes the specified function (in this case a `doWait(5)`) this function is defined just below and waits for 5 s.

The watch function is started by a `watch doors look_for_door(-90)` specifying the function to watch and gives the watch a name `doors`.

The drive command `fwd left dist :((dd + distSoFar) > dist)` is found near the end of the script. This drive command may be interrupted by the `skipcall`, and as the variables in the stop criteria `distSoFar` (distance driven within this drive command) are not maintained between the calls the variable `dd` holds the distance driven from before the last interruption.

Before the end the watch is closed by the `unwatch doors` command.

This functionality should enable some error recovery, like backing off if the robot is stuck in obstacles. A number of the tests have been run with a simple speed monitor, that – if speed is zero for 6 seconds – would try a reverse drive to a position visited a few metres back. The function did not succeed in the few cases where it was activated for two reasons: when the robot is stuck in the high grass the laser scanner detects the grass as near obstacles and shuts off the drive control (a security measure). In other situations the robot simply was stuck (unable to move by own power). In both situations a manual assistance was needed.

Another possibility is progress monitoring to allow a change in strategy if the primary fails. Such behaviour is not tested and the used scripts have therefore been of the type: succeed or die.

6.3.6 Support functions

A number of support functions are available in the script language. These include many of the normal math functions like `sin(a)`, `atan2(y,x)`, `min(a,b)` and `sqrt(v)` and a number of functions that relates to robot navigation. Many of these support functions just reduce the number of lines in the script while others provide access to more complex data structures. A few of the support functions related to robot navigation are described below (the full set of support functions is listed in appendix B.

Tail rope

The odometry position for the last about 100 m are stored as a sort of tail rope. It marks positions that (once) were traversable and describes the route taken. This tail rope is already used to maintain a system variable `histH` for the 'historic' heading of the last 5 m excluding the most current 1 m. This heading is used when evaluating alternative routes in the obstacle avoidance

planning.

A more general set of functions are available to use these historic data as listed below

```
calcPoseAtTime(time)
calcPoseAtDist(near)
calcPoseFitAtDist(near, far),
```

where `calcPoseAtTime(time)` returns the historic posture a specific number of seconds ago, the returned posture is – as for all functions starting with `calc` – stored in the variables `calcX`, `calcY` and `calcH`, respectively. The function itself returns `true` if such a posture is available.

The `calcPoseAtDist(near)` returns the tail rope posture at this (direct) distance from the current position.

The `calcPoseFitAtDist(near, far)` uses all the historic positions from `near` to `far` from the current robot position. These positions are fitted to a straight line. The closest position on this line to the posture at the `near` distance is returned in the three posture variables. The function itself returns the residual variance from the line fitting.

One of the uses of this functionality is when the robot reaches a side road or a junction and needs a position and direction where the road ends before the junction. This position and direction may be needed when calculating a direction out of the junction.

An example of this use is shown below

```
...
// get general direction of road now
calcPoseFitAtDist(0, 25)
// follow left side into junction
fwd left 75 :(7.0 < abs(
     distToPoseLineSigned(poseX, poseY, calcX, calcY, calcH)))
// get reference close to end of road
calcPoseFitAtDist(20, 40)
// add a vector in the right direction
calcAddRel(75.0, 150.0)
...,
```

an example of this manoeuvre is shown in Fig. 6.3.

While on the road and at some (less than 75 m) distance before the junction a call to `calcPoseFitAtDist (0, 25)` is made (at (1)) to mark a line in the road direction. The next drive command using the stop condition `(7.0 < abs(distToPoseLineSigned(...)))` takes the robot into the junction and until 7 m from the road direction line (at (2)). This is a rather fixated position

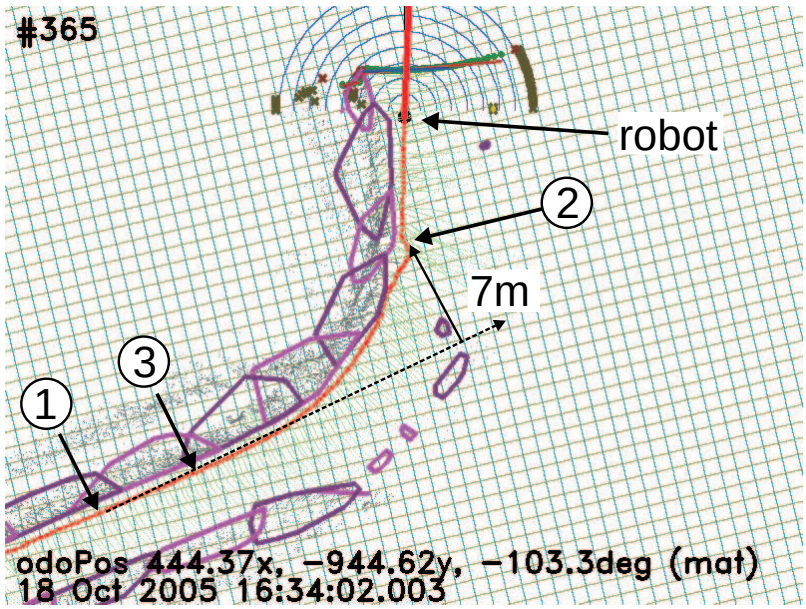


Fig. 6.3: Determination of where a road ends and a junction starts may be a bit tricky. At (1) the road direction line is stored. The left road edge is then followed until the robot is 7 m away from the stored road line (at (2)), this is a known position a short distance into the junction, and a road end posture is then estimated from odometry history 20 m back as an on-road reference point (at (3)). From this a relative position is calculated towards the exit road.

relative to the junction, and from here a position and direction of the road just before the junction is calculated by `calcPoseFitAtDist(20, 40)`, Then this position 20 m back (at (3)) is used to calculate a direction towards the right exit road in the junction.

GPS heading

The position accuracy of the GPS is about 5 m at the best, so a historic position about 20 m from current position may be usable to get a reasonable GPS heading. A distance of up to more than 50 m may be usable if the odometry errors can be assumed to be small. The heading is calculates from UTM coordinates as

$$\theta_u = \angle(\mathbf{U} - \mathbf{U}_d) + \theta_o - \angle(\mathbf{O} - \mathbf{O}_d), \tag{6.1}$$

where $\mathbf{U} = [E, N]^T$ is the current UTM coordinate and $\mathbf{U}_d$ is the UTM coordinate at an appropriate distance d. Similarly $\mathbf{O} = [x, y]^T$ is the current odometry position and $\mathbf{O}_d$ the odometry position at distance d. θ_o is the current heading in odometry coordinates.

The angle θ_u is oriented with East as zero and positive angles counter-

clockwise. This angle may be transformed to a compass angle (in radians) by

$$\theta_{\text{compass}} = \frac{\pi}{2} - \theta_u. \tag{6.2}$$

6.4 Mission assignment - operator interface

Conceptually an operator interface is needed to provide the robot with a mission, and depending on the mission there may be a need for a more complex dialog between the operator (or user) and the robot. In a real world situation with one or more robots deployed, a mission monitoring function would be needed to monitor the robot progress for efficient allocation of new missions.

For the end user and especially for the more technical fault finding and debugging purposes an operator interface and monitoring ability is needed for all parts of the robot. Especially for research experiments an operator interface is needed to allow easy change of parameters and intensive monitoring of performance. A comprehensive operator interface is therefore available that supports interface to all servers developed for this project. The interfaces are all text (XML) based especially to ease test and debugging.

The mission assignment to the navigation scheduler is an online operator function, replacing the mission decision functionality.

A mission is normally loaded as a full script file using an online command like

```
<planSet load="scripr.mmr"/>
```

where `planSet` is the command identifier to make changes in the navigation sequencer, and `scripr.mmr` is the filename of the file with the mission script.

The mission can alternatively be added line by line using the operator interface command

```
<planSet add="command"/>
```

where `command` is the command line to be added.

The robot can thus be controlled by adding one line at a time. A separate function allows the newly added line to be executed immediately, like

```
<planSet goto="command"/>
```

that adds the `command`, skips the current command and proceeds with the added command.

The shown user interface commands are packed in XML brackets, as they describe the command message format to the behaviour server.

Fig. 6.4: The primary test route. The start position is on a rather long asphalt road. The total route is about 3 km. The map background is from Google Earth.

6.5 Mission planning

The mission planning is primarily prepared from a topological map of the area in which to navigate, supplemented by a more metric-based crossing of the junctions. The primary test route is shown in Fig. 6.4. It starts on a rather long asphalt road with almost no side roads. After this, in front of the hunting lodge, the asphalt road ends and the robot has to cross a gravelled area, and subsequently a number of forks and junctions need to be passed before the end position is reached.

The navigation mission planning is first to select which roads can be handled by following one side of the road. Then the detection of junctions may be needed, and finally a method must be found to get from one road to the next across the junction.

The major part of the script can be prepared from this map, as

`fwd left 1740` (asphalt road towards hunting lodge),
detect road end,
cross open square,
`fwd left 300` (Keep left while passing two sideroads to the right),
`fwd right 280` (keep right until the 5-road junction),
detect junction,
cross junction,
`fwd left 280` (to get to the final destination).

All the roads are assumed to be handled by following one of the road lines. The two junctions that can not be crossed by following a road edge left or right need further planning.

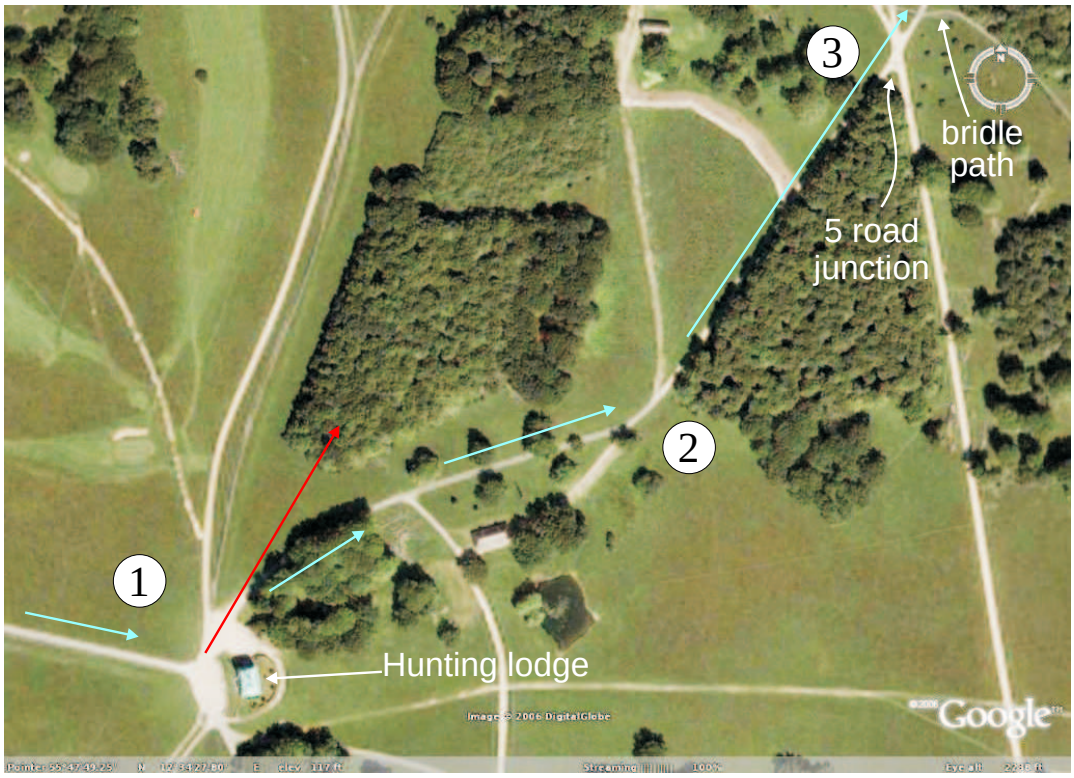


Fig. 6.5: The primary test route detail from the hunting lodge to the end. The difficult areas are the rough roads starting at (2) and the junctions (1) and (3). The background image is from Google Earth.

A slightly more detailed map is shown in Fig. 6.5. In the first junction (1) the exit road is the second to the left, and it may be possible just to follow the left side until the first road is detected and then return to the square and follow the left edge to second road. This method may work, but it is not the optimal solution. Therefore the plan is to detect when the first road ends and aim from here towards the exit road (the red arrow on Fig. 6.5.

The detection of the road end could be from a GPS position (obtainable directly from the Google map) this would most likely be found with an accuracy of about 5 m as it is in an open area. The road width could be used, but as the road width reduces in quality when the distance to the road edge is more than about 5 m away from the robot, a consistent detection would be rather doubtful.

A more precise road end could be found using the method from Fig. 6.3, where a road direction line is used to detect when the robot has departed from the road. When the road end is detected a new direction is calculated based on the road end. A position in the right direction is then followed until the new road is detected. This last method has proved to be a reliable way to cross the

junction.

At the second junction (3) it may be possible to follow the direction of the road before the junction and cross both the junction and the following bridle path. A GPS point is set to trigger a road direction line generation using a `calcPoseFitAtDist(0, 20)` from position history 20 m back. A further 100 m along this line a position is found, and the robot is set to aim for this point. The road sensing should ensure that the robot follows the road, even if the aiming point is a bit off the road. After crossing the junction (and the bridle path) a road edge can be followed all the way to the goal point.

The potential script could then be prepared as

```
fwd left 1740 :(25 > distToGps(347716.7, 6183490.1),
calcPoseFitAtDist(0, 25) (to get direction line of road),
fwd left 75 :(7.0 < abs(distToPoseLineSigned(
              poseX, poseY, calcX, calcY, calcH))),
calcPoseFitAtDist(12, 25) (gets better road direction line),
calcAddRel(75.0, 150.0) (in direction of exit road),
gotowaypoint odo calcx calcy :((roadLeftQual > 0.78) and
              (roadLeftDist < 2.7) and
              (distsofar > 10)) (until road edge is detected),
fwd left 300 (Keep left passing two sideroads to the right),
fwd right 280 :(25 > distToGps(348223.0, 6183901.0)),
calcPoseFitAtDist(0, 20),
calcAddRel(100, 0), (across the second junction)
gotowaypoint odo calcx calcy :(endPoseDist < 45),
fwd left 350 :  (30 > distToGps(348485.0, 6184198.0)
```

The script fails at two points, as the map did not tell the full story. The most severe error was the five road junction (at (3) in Fig. 6.5), where the crossing road is at a higher level than the followed road. While climbing onto the crossing road the laser scanner detects parts of the road at a very small angle, as a consequence parts of the road are seen as obstacles resulting in a failure to find the road (the entry is partially blocked by the erroneously detected obstacles). It turns out that following the road to the left onto the crossing road, followed by a turn in the direction of the other road produces less false obstacles and the road is crossed more successfully.

The rough, gravelled road starting at (2) in Fig. 6.5 has rough edges, that produces low quality road edges. The vision sensor improves the situation, but the driving is very irregular, and the mission fails occasionally when following the left or right road edges. However the gravelled road has a road profile that is detected with high quality by the centre road line, and the route is traversed more smoothly when following this line.

Additionally the drive speed and other values need to be initialized in the script. The full script used for the route can be seen at the end of appendix B.

The result is that a navigation script is not likely to succeed unrehearsed. After a first rehearsal the difficult parts of the road may need an adjusted navigation script to succeed.

6.6 Summary

In many situations the mission planning will not allow unrehearsed scripting for a navigation path. Especially at rough roads and junctions, that at places may challenge the capabilities of the robot, may need careful scripting.

The set of command provided is capable of implementing the needed command script.

The implemented watch functionality has the potential of detecting more complicated situations, these could include detection of side roads and mission threatening situations. Mission threatening situations could be: getting stuck in the rough grass or taking a wrong turn. A detection of such a situation could then trigger a skip to a recovery manoeuvre. The usefulness of this functionality has not yet been proved.

The navigation script functionality has been tested comprehensively and is able to implement navigation missions successfully.

Chapter 7

Software architecture

7.1 Introduction

The amount of software needed for a functional autonomous robot is significant, at least if the robot should be capable of working at a level, where experiments with navigation and mission management are relevant subjects.

Software reuse is an obvious solution and when looking at the internet a large number of open software solutions are available for e.g. motor control, stereo vision, GPS interface and extended Kalman filter. Some of the solutions may include reusable parts, but the work needed to understand and adapt the solutions may be almost as extensive as a recreation from theory.

When reusing software efficiently it should neither be required to understand the programming structure nor the coding details, the functionality and the interface requirements should be sufficient. For this to work a common structure or infrastructure is needed.

The software structure needed for navigating mobile robots is the subject of this chapter. The chapter discusses the available solutions and proposes yet another. The purposed software architecture fits the requirements of a research platform, which allows easy incorporation of new solutions, as well as separate maintenance of the individual components.

7.2 Related work

The three best known component-based mobile robot oriented solutions are the Player by Vaughan et al. (27-31 Oct. 2003), Carnegie Melon Robot Navigation Toolkit (CARMEN) by Montemerlo et al. (2003), and Orca by Brooks et al. (2005). All are based on open source and is available for download from 'http://sourceforge.net'.

The component structure in Orca implements clients and servers and lets

these communicate in modes like client pull and server push. The communication subject on each connection is well defined. The implementation builds on software reuse, partially by using a series of available libraries and partially by allowing reuse of the produced components. It is not widely used and the number of components is small.

The Player solution is not strictly component based as all sensors and controls are integrated into one server, this server then allows plugins and communication with external components. Much of the bulk and real-time demanding communication is thus embedded in the server and shared to the plugins, increasing the performance of these issues.

The CARMEN toolkit is component-based and communicates using an IPC[1] (Simmons (2005)) centralised communication that supports the applicable communication modes between components. Separate components are available for a number of existing robot platforms and sensors (including laser scanner and webcam). The sensor components are typically hardware abstraction components that adapt the sensor data flow to the IPC communication framework.

7.3 Software architecture requirements

University research projects are often divided into individual projects or projects in small groups. Each project has a limited, and often very short, time frame in which results have to be produced. Especially in robotics projects a maintained software base is needed if projects are to be undertaken at more than the lowest levels (like sensor data extraction and drive controlled directly from one sensor). Testing a complicated sensor, an alternative behaviour algorithm, a SLAM[2] solution or other advanced functionality requires either a large project – where most of the resources are spent on the basic functionality – or an established high-level infrastructure on which the new functionality can be built.

The new functionalities built in university projects are mostly discarded after the reports have been evaluated, some of them for good reasons, but others should have been added to the available infrastructure for new projects. To make this possible – without a major software maintenance effort – the software architecture must support such an incremental development. Preferably the software should be sharable among universities and other interested parties, for increased functionality and reliability (the more people that use the software the more reliable it gets).

The main requirements to robot software architecture are thus:

[1] IPC: Inter Process Communication
[2] SLAM: Simultaneous Mapping and Localization

- Maintainable in separate units. It should be possible to develop and maintain a unit of the robot software independent of other software units. The development and maintenance of a unit should not require more than a functional knowledge of the other units to which it is to be connected.

- The communication between units must be open, simple and extensible. Open to allow others to continue maintenance when needed. Simple to simplify development and debugging as most software developed for research are prototypes and thus makes ease of debugging a vital theme. Extensible to allow addition of new features without affecting communication partners that do not need the extension.

- Each component that fulfils an exclusive function on the robot should be expandable. An exclusive component occupies a vital resource or function in the system; this could be a sensor (eg laser scanner or camera), a manipulator or actuator, a behaviour server or a mapping function. When a new extension is to be tested for such a component it should be possible to do so without a need of a total replacement of the component and without overtaking the maintenance responsibility of the basic component.

- Data communication with high bandwidth devices (like cameras or laser scanners) should be considered. Data analyses of data from such devices should be possible close to the data source.

7.4 Design decisions

The component structure in Orca was investigated but was not adopted, primarily for three reasons. The communication structure seemed too complicated (too much overhead for high data rate connections and too limited in data content for each connection – resulting in too many connections). The extensive library dependency gave implementation difficulties in our Linux distribution. The available components were not sufficiently attractive to overcome the other limitations.

The Player structure was too monolithic and the work involved to adapt to our platform seemed discouraging.

The CARMEN robot toolkit has the benefit and the disadvantage of the centralised communication structure. The communication with high bandwidth devices has to pass this communication system, and there is no possibility (within the toolkit limits) to reduce the data rate at the source by attaching the data analysis to the data source. Faster hardware is a solution, but the faster hardware may be better spent on increased functionality.

The decision was therefore to design yet another robot framework, the benefits being that the available software for our robot platform could be incorporated as is. The available software includes a robot hardware abstraction layer and a drive scheduler. The software architecture could be shaped to the needs as envisaged in the above requirements.

The proposed software architecture is illustrated in Fig. 7.1.

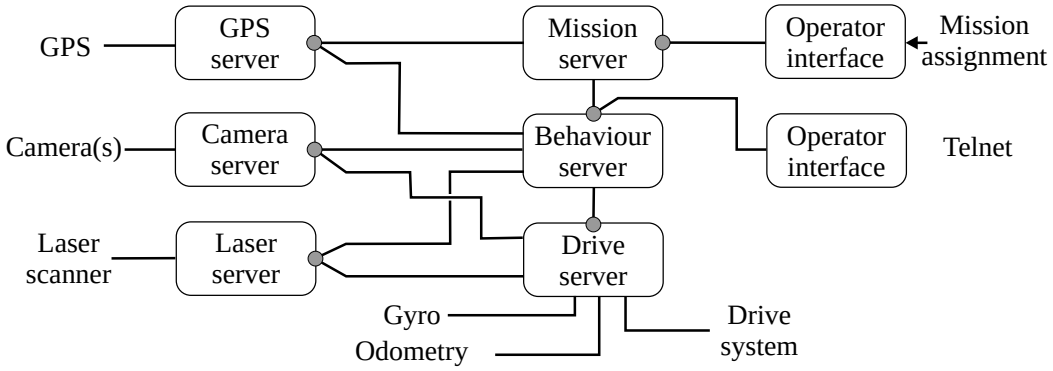


Fig. 7.1: The software architecture is based on a number of server functions that allows connection from clients at points marked with a circle. The message format is text (XML) based and allows thus simple monitoring and control of all servers from tools like eg TELNET

The architecture is based on a number of servers where client connections can be attached. This architecture, as well as the basic communication methods – server-push and client-pull – is taken from the Orca terminology. The communication media is selected to be socket-based TCP/IP without a central communication manager. A server occupies one port number only and all the services provided by the server are communicated through this port. The server may be attached to other servers through client connections.

The communication data formatting is selected to follow a subset of the XML (Extensible Markup Language) defined in the World Wide Web Consordium (2004) recommendation. The interface standard between a client and a server is therefore primarily based on an agreement between a client and a server on the used tags and attributes in the XML formatted communication. XML is selected as it is well defined, and it remains compatible even if extended, it is text based allowing for interface debugging by use of simple tools – like TELNET. Additionally a number of parsers and coders exist in different programming languages.

Transfer of large data structures (eg a camera image) over a socket connection is relatively slow – XML coded or not – compared to data transfer internally in one application or by using shared memory. An internal data transfer rate may be up to 4 bytes for each CPU clock cycle (on a 32 bit pro-

cessor). Transfer of the same data on a socket connection is often at least 10–40 times slower.

A server handling large data structures – like a camera server or a laser scanner server – should therefore include as much data reduction processing as possible before delivering data to a client. That is, the server should hold the data analysis functionality so that only the result is transferred to the client. Much of the new data analysis functionality should therefore preferably be built into the server directly. To do this without sacrificing separate maintenance of reused code and new functionality, the new functionality should be added to the server on a plugin basis.

The software architecture is therefore, to some degree, a mixture of the three candidate software architectures discussed above: a component structure like in CARMEN, a plugin capability like in Player and a direct communication structure like in Orca.

7.5 Communication

The data transfer is illustrated in Fig. 7.2, where posture data in the drive server is to be transferred to the behaviour server. The data structure is coded

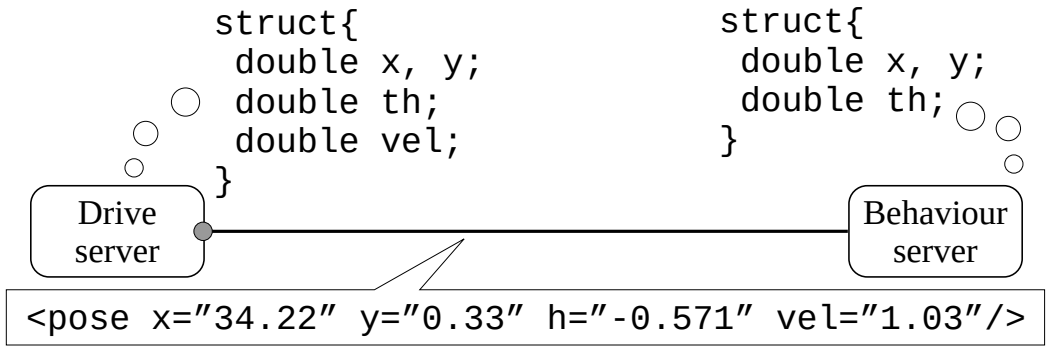


Fig. 7.2: A data structure is converted to XML-based text format on the communication line and converted back at the client end.

as an XML tag `pose` with the data values as attributes. In the behaviour server the posture is transferred back to a similar data structure. The extra velocity field `vel` may be an extension to the data structure in the drive server, but the behaviour server may not need to be modified to understand this extended data structure.

The data flow is always controlled from the client. A client can request data from a server as a one time service, a client pull data transfer like

```
<gmkGet device="1" width="0.025"/>,
```

which is a request to the camera server to analyse an image from camera device one for standard guidemarks with a block width of 2.5 cm. The reply from the camera server could be

```
<gmkGet gmkCnt="1">
<gmk code="0007533" cam="left" device="1">
<timeofday tod="1154682383.782351"/>
<pos3d name="gmkPosition" x="2.0302" y="0.5201" z="-0.1500"/>
<rot3d name="gmkRotation" Omega="1.56235" Phi="-0.22109"
          Kappa="-2.92682"/>
</gmk>
</gmkGet>.
```

An alternative communication method is server push, where the client requests the server to repeatedly send a data series, like

```
<push t=0.5 cmd='gmkGet device="1" width="0.025"'/>,
```

which when sent to the camera server will trigger a guidemark analysis every 0.5 s from camera device 1, and the data found is returned in the same format as before.

An alternative server push method is to transfer data every time an event happens, eg every time there is a new image from a camera, then perform some operation and return the result, eg

```
<camPush i=3 cmd='gmkGet device="1" width="0.025"'/>,
```

where every third image (`i=3`) is analysed and the data returned to the client as before.

The servers may return data to an operator interface and may even be used as a stand-alone application for some interface, and operated from a text-based interface to the server socket.

7.6 Component structure

All components basically have the same structure, a server port, some resources and a set of function adding modules. The modules may include connections to devices or to other servers.

The server itself holds the basic server functionality. The basic functionality includes a server port and the ability to load modules. The server port handles client connections and a few commands like `name` and `time`. The module loader is used to add the specific functionality to the server. One or more of these modules provide the basic resource needed for the specific functionality of the

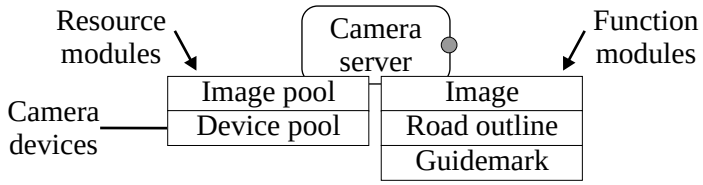


Fig. 7.3: A camera server with a number of attached modules. One module handles the interface to camera devices in a device pool, additionally an image pool module with reusable image buffers is available; both are made available to other modules in the server as server resources. The additional function modules then utilise these resources.

server. These resources are then provided as server resources available to all other modules.

The basic construction is illustrated in Fig. 7.3 for the camera server. Each module adds some functionality and makes its services available for the clients, and the resource modules further makes the resource functionality available directly to other modules. The functionality is accessed through a unique keyword or a set of keywords. The server directs an XML tag with a matching keyword to the module handling that keyword, and provides information about the client requesting the function.

The modules can be added as permanent modules linked into the server code or be added as plugin modules. Resource modules are added as permanent modules to provide common resources of the server. Plugin modules can presently be added when starting the server application only, but should be extended with online module load and unload functionality, in order to allow testing without restarting the server component.

As the components are applications in their own right and controllable from a text-based interface, an online help system is made available. The server can provide a list of modules and the keywords handled by each module. The list in table 7.1 is a list of modules and the handled keywords, which is returned by the camera server using the function `<sysGet functions/>`

Table 7.1: An example of modules and function keywords loaded by the camera server.

Module	Function keywords (name and version)
server	`push q sysGet name time help (camera_server-1.25)`
camera pool	`camGet camSet camPush camsGet (camControl-1.22)`
image	`imageGet (image-1.22)`
image pool	`poolGet poolList poolSet poolPush (imagePool-1.2)`
road outline	`pathGet (imagePathFinder-1.25)` and
guidemark	`gmkGet gmkSet (imageGuidemarkFinder-1.25)`,

The first line in table 7.1 (including `sysGet`) is the function keywords handled by the server directly, the rest of the lines are the loaded modules including the technical name of the module and its version number.

The available parameters for each function may be listed using an online help function, like `<gmkget help/>`.

7.6.1 Client connections

A server module may include a client connection to some other server component. Client modules are available for the most common data structures. Each client module receives and unpacks one or more XML formatted data structures into a C++ structure usable by the server module. The client module includes trigger functions that are called when a full data structure is received.

7.6.2 Mission monitoring

The operator interface component is the only non-server component. The purpose is to be able to monitor the activity in each of the server components and present the information in an appropriate form.

The most simple operator interface is the text-based network tool called TELNET, from this tool it is possible to query any of the servers for functionality, but the reply may be too voluminous to be meaningful. A camera image provided in text form will in most cases be difficult to interpret.

An operator interface that provides a graphical interpretation of most of the more complex data structures is therefore needed. The purpose is both to monitor the progress of the primary functionality and to make adjustments to the available parameters, eg to load a mission for the robot.

Many of the illustrations used in this thesis are directly produced by the operator interface.

Each server is a stand-alone application with communication to the relevant other servers and resources, there is thus no central function that provides an overview of the entire network, nor the amount of inter-component communication. Each server must be monitored separately.

7.7 Simulation

Simulation is an important function for any robot research project. In the available set of components simulation is handled by creating a simulated device or interface in the appropriate server. This simulated device then gets its data from a common simulator system. At present a simulator is available which may simulate the robot in a 2D environment and from this provide data for most of the robot sensors, including odometry and laser scanner.

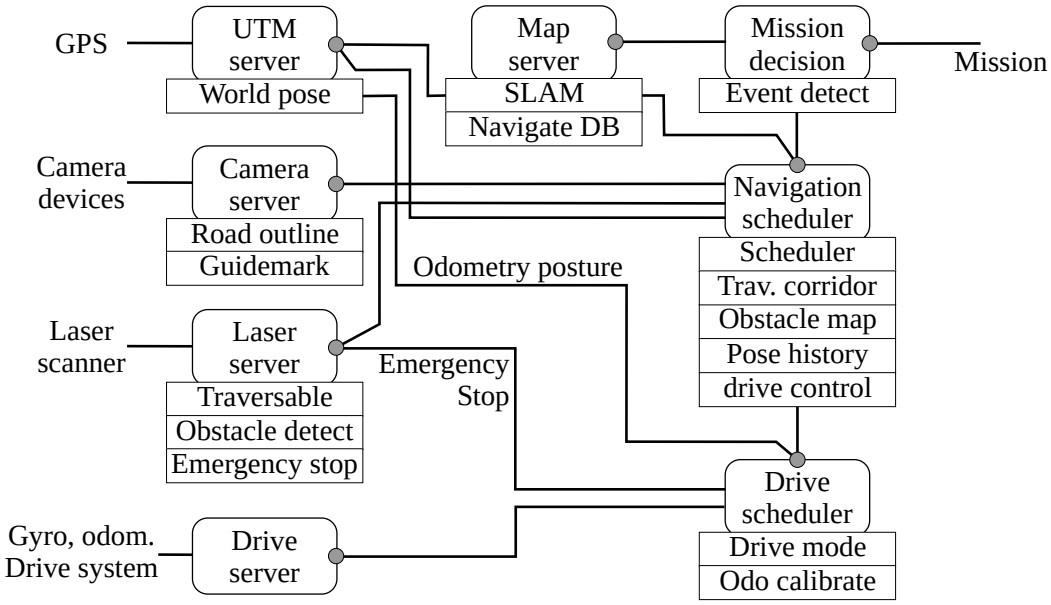


Fig. 7.4: A complete component based robot software structure based on the task breakdown in Fig. 2.2. The servers are the rounded squares. The squares below are server modules. Client connections are attached to a server at the small circles.

For this navigation project the simulation is primarily performed by replay using log-files from real experiments, replacing the drive scheduler server, the laser scanner server and the camera sensor. The camera server supports simulated camera devices where the images are loaded as if the source was the simulated device.

7.8 Full component structure

The work breakdown presented in chapter 2 and shown in Fig. 2.2 could be implemented in the presented software structure as shown in Fig. 7.4.

The main information flow between the components is described in the following.

The drive scheduler server produces an odometry-based posture, which is used by the UTM server to produce a world posture in combination with the GPS.

In the map server the global map is maintained based on traversable corridor and possibly obstacles from the navigation scheduler server.

The navigation database in the map server is used by the mission decision server to select the appropriate navigation path for the top priority mission. The selected navigation path sequence is sent to the navigation scheduler.

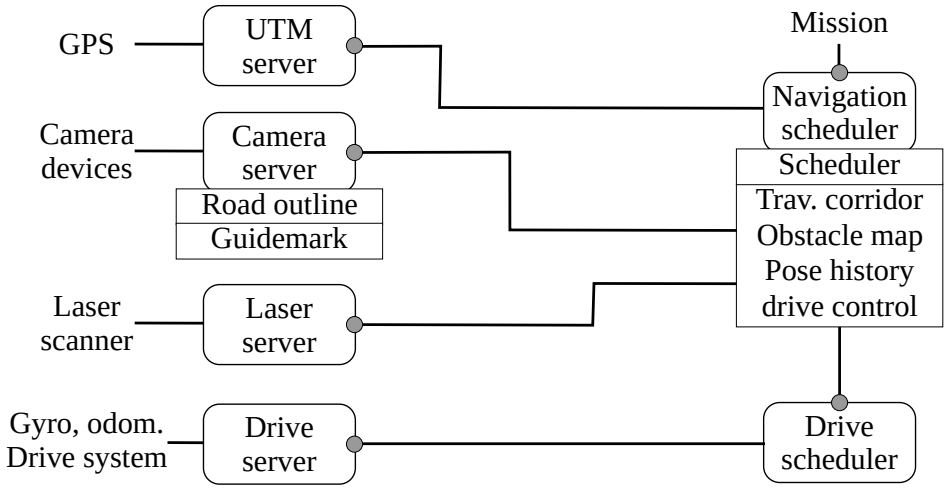


Fig. 7.5: The implemented structure does not follow the described software architecture for all components – compare with the desired structure in Fig. 7.4. The camera server and the navigation scheduler use the component structure only.

The navigation scheduler plans a desired short term exit posture based on a traversable corridor. The traversable corridor is obtained from a sensor fusion of road outline from the camera server and the traversable segments from the laser scanner server. The navigation scheduler then plans a path to the desired short term destination and transfers a sequence of drive commands to the drive scheduler.

The drive scheduler server executes the desired drive commands by controlling the drive system.

This component structure is only partially implemented as described in the next section.

7.9 Results

The component structure of the implemented and tested system is not expanded to the level shown in Fig. 7.4 for a number of reasons. The actually implemented structure is as shown in Fig. 7.5. The drive server and the drive scheduler are existing components, they are well-proved socked-based servers, but they do not comply with the described architecture.

A laser scanner component is available but is not yet implemented in the tested system; the laser scanner functionality is therefore transferred to the navigation scheduler.

The camera server is implemented following the described architecture. The navigation scheduler is partially implemented as described, except for the mod-

ule structure where the module separation into resource modules and function modules is missing.

The GPS server is not component based as described and has no world posture functionality.

The map server and the mission decision components are not implemented.

The concept of having an established infrastructure that is divided into functional components is a concept that divides the complex functionality into manageable entities. Adding the text-based interface makes the components easy to grasp, and with the functional plugin modules makes the effort required for functional extensions relatively simple to implement.

The implementation of resource modules are not sufficiently standardised for easy implementation.

The separated maintenance concept works for the components, as the use of XML makes new and old components compatible over many extensions. The same is not true for the plugin modules. A few updates to a shared resource may have fatal consequences when attempting to load a module with a different expectation of the shared resource. This is a foreseeable shortcoming and imposes a stronger requirement of the stability of shared resources.

7.10 Summary

A software architecture is proposed that emphasises software reuse and the ability to introduce new functionality with only a short introduction to the architecture and components. The built components – especially the camera and laser scanner components – have proved these design objectives in a number of projects.

The implemented communication protocol based on XML is flexible, extensible and easy to understand and allows components to be written in many programming languages and on different platforms. The interface protocol relaxes the need for strict version dependency across components.

The component structure is not fully implemented and especially the resource sharing plugin modules have not yet found their final form. It is believed however that the architecture is well suited for the task, it suggests a flexible but strict structure, and is sufficiently pragmatic and flexible to allow interface to components that do not follow the same structure.

Chapter 8

Results and discussion

8.1 Introduction

This chapter presents some of the results obtained in the test area not already covered in the previous chapters.

The results presented in this chapter is from a test drive performed 28 July 2006, where the results elsewhere cover a test period of almost a year – from October 2005 to July 2006. The early data being without the vision functionality, as this was not implemented at the time.

8.1.1 Test route

GPS is used at a few locations only, but the GPS position is logged at all times. The interesting part of the test trail from 28 July 2006 is shown in Fig. 8.1 based on this GPS log-file.

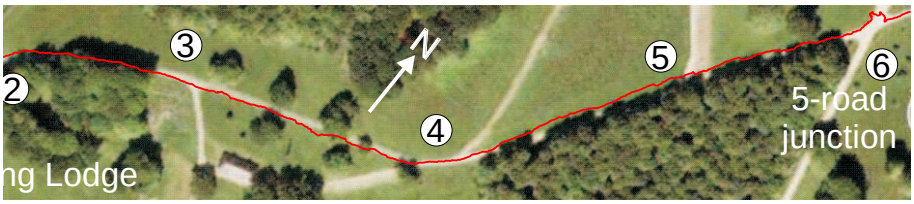


Fig. 8.1: Part of the test route plotted from the GPS log-file. The GPS is almost sufficiently accurate to show the track inside the visible road on this satellite image (the robot actually stayed on the road all the way).

8.2 Road line quality

The road edge line quality is shown in Fig. 8.2 for a part of the route shown in Fig. 8.1.

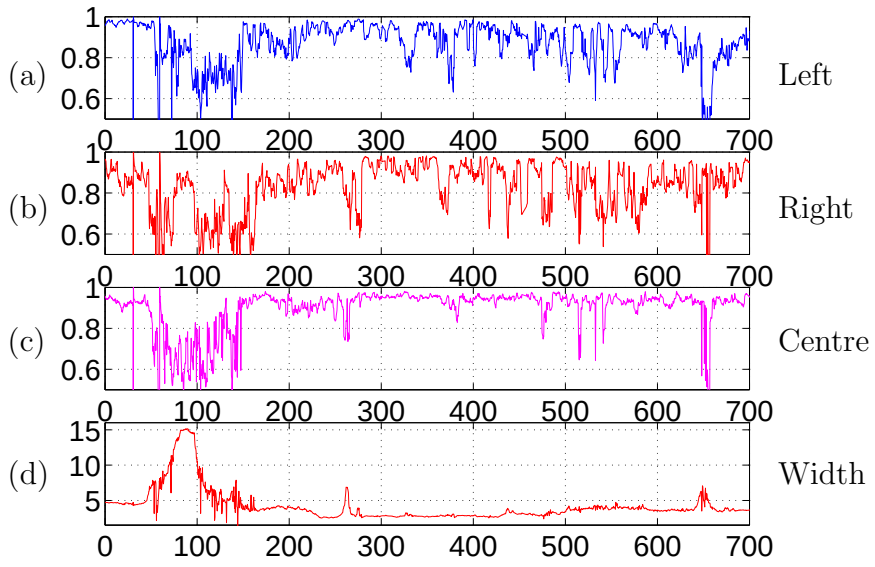


Fig. 8.2: The curves show the estimation quality for the left (a), right (b) and centre (c) road line, respectively. The estimated road width is shown in (d). The x-axis is the travelled distance in metre and the road edge quality is in the range [0,1]. The distance is – with reference to Fig. 8.1 – from just before the hunting lodge (1) to just past the fork at (5).

The first 45 m is on the main asphalt road where the robot drives at the left side of the road, the left road quality is high (almost 1.0 – top graph), the right road line quality is lower (about 0.9 – graph (b)) as it is further away, the road centre line quality (graph (c)) is well defined on the asphalt road with a high quality.

After the 45 m the robot enters the open area in front of the hunting lodge. The quality drops for all three lines. When the road width is estimated at its widest (about 15 m) the road line quality is rather high (about 0.85). The quality is the residual from the line fitting, and as the road width estimate constantly is as wide as the laser scanner range permits, the residual from the line fitting is low. The road edge line quality can thus not alone be taken as an evidence of a road line being detected; the road line quality must be combined with the detection distance from the robot.

At 100 m the estimated road width is narrowing as the robot regains contact with the road edges. The road edge quality stays low until about 150 m. The area (at (2) in Fig. 8.1) is very flat and the roadside is difficult to classify by

the laser scanner.

At about 150 m the road changes to asphalt and especially the centre of the road is now tracked with a high quality. At 275 m (at (3) in Fig. 8.1) the side road to the right makes the right and centre road qualities drop, but the left road edge quality level is maintained.

After another 200 m at about 350 m the road type changes back to gravel. The road profile stays rather convex and the road centre line quality remains high for the rest of the route – until the last junction (5) at 650 m.

8.3 Excessive roll

Excessive roll makes the road detection in one side of the robot much closer. Occasionally such close detections may violate the obstacle detection assumptions. Figure 8.3 shows a situation where the robot enters a small pit resulting in a roll to the right. The laser scanner detection from the road is thus much closer on the right side.

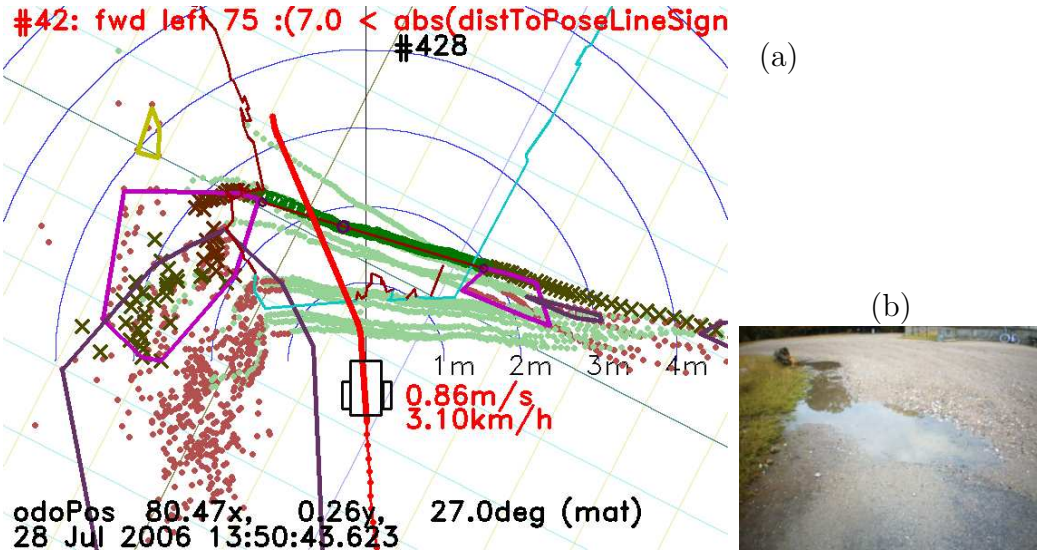


Fig. 8.3: When entering the open area in front of the hunting lodge at (1) in Fig. 8.1 there is a pit filled with water on the route as shown in (b). When the robot enters this pit the robot makes a roll (about 8°) to the right, and thus the laser scanner detects the right side of the road at a shorter range – as shown in (a). The traversable laser scanner measurements are in green whereas the nontraversable are in a darker brown.

Where the detections are closer than 1.2 m in the x-direction the h_{sup} criterion in the raw height feature extraction in section 3.5.4 is triggered, assuming that the detected object is 0.2 m or more above ground level.

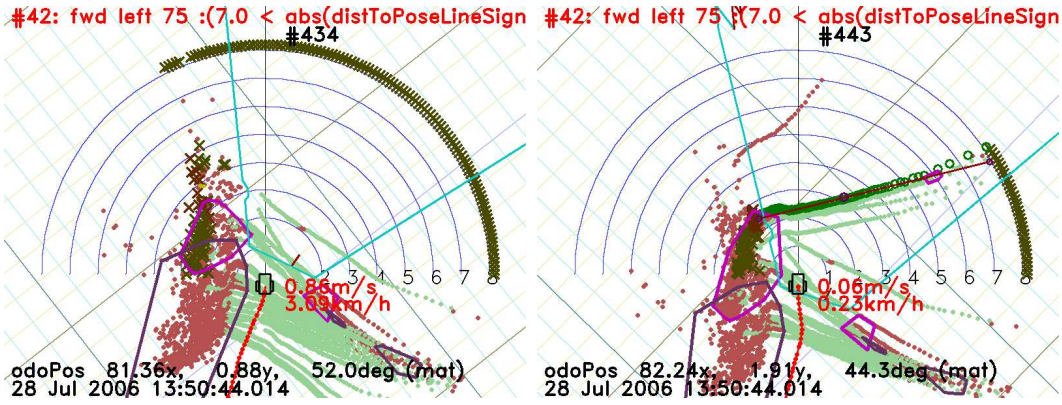
As a consequence this part of the road is classified as nontraversable and included as an obstacle as shown in Figure 8.3(a).

The erroneous classification of the road does not affect the classification ahead, and thus the continued drive is unaffected. This generation of erroneous obstacles may be a problem if they are to be used for mapping purposes.

The situation may have been be resolved correctly if the robot were equipped with a roll sensor. Alternatively the height criterion could be enhanced to allow some degree of roll, eg to allow 10° roll as is more than the 8° experienced in Fig. 8.3.

8.4 Excessive tilt

When the robot enters a pit this may result in an excessive roll as described in the previous section, when the robot is leaving the pit an excessive tilt may be the result. This situation is illustrated in Fig. 8.4(a).



(a) (b)

Fig. 8.4: When entering and leaving a pit the robot tilt may influence the measurements excessively. In this image – a few scans after the situation in Fig. 8.3(a) – the tilt is so high that the laser scanner does not detect the road before maximum range, and hence no measurements are classified as traversable (a). In (b) the robot has stopped as no navigation corridor is available.

The figure shows that the laser scanner does not detect the road surface at all; all measurements – apart from the obstacles to the left – are at maximum range (8 m). The tilt of the robot must therefore be more than 5.5° when leaving the pit.

The situation produces no traversable road segments, and thus a navigation corridor is not established. The robot continues to follow the remaining part of the last planned route, ie less than 1 m, after this distance the robot stops at the position shown in Fig. 8.4(b). After a few seconds the scans with no

traversable segments are timed out (as of section 3.6.2 $T_{\text{fade}} = 4.5\,\text{s}$) and the robot continues.

The vision road outline shows (as expected) a free path, but this is not trusted if no corridor is generated based on laser scanner data. The situation is not critical unless the robot stops in a situation where no traversable segment is found (robot tilt above 5.5°), in such cases the robot is stuck. A situation where the robot reaches a cliff edge or a steep downhill edge would show similar detections. A tilt sensor could have been used to resolve this situation.

8.5 Open areas

When driving in open traversable areas the road edge lines are estimated close to the maximum range of the laser scanner as shown in Fig. 8.5.

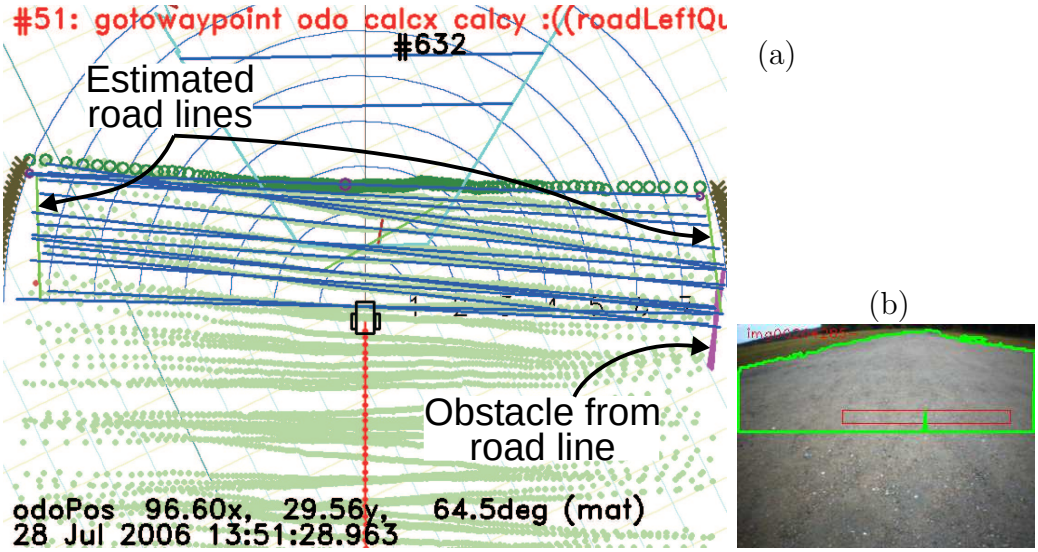


Fig. 8.5: The open gravelled area in front of the hunting lodge. The full detection range of the laser scanner is classified as traversable (green dots and circles in (a)). The camera view can be seen in (b).

The estimated road edges left and right, are both at maximum range of the laser scanner. The residual error from the line fit of the left and right road lines is small and hence the detection quality is high. The quality of the right edge line is sufficiently high ($L_q > 0.9$ as from section 5.6) to be added as an obstacle. Both the high quality and the obstacles are undesirable and could be avoided if road edges close to the maximum range of the laser scanner were prohibited from being used in the road edge calculation.

8.6 Flat roadsides

The road leaving the open area at the hunting lodge (at point (2) in Fig. 8.1) constitutes a problem for the laser scanner classifier as illustrated in Fig. 8.6.

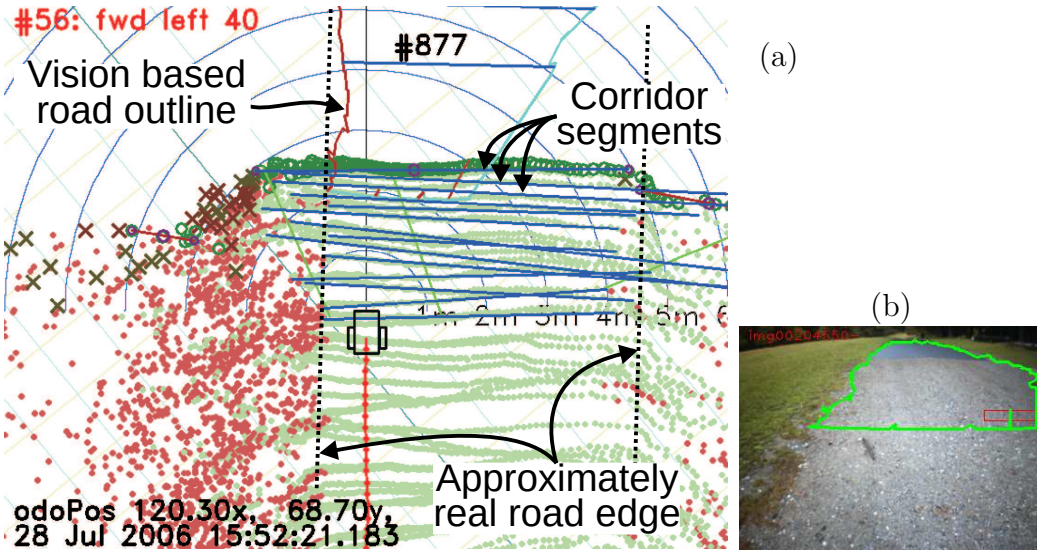


Fig. 8.6: Data from the area marked (2) in Fig. 8.1. The blue lines in front of the robot are the traversable segments in the used corridor. The closest part of the vision-based road outline is shown at the top of the image. The vision source image is shown in (b) with the estimated road outline.

At areas where the roadsides are flat and almost in the same plane as the road, the laser scanner classifier tends to include parts of the roadside as traversable road. In Fig. 8.6 the traversable segments (the blue lines) are extended up to 1 m into the left roadside, compared with the vision-based road outline.

The ragged edge detection gives a poor edge line quality, and when the end points of the traversable segments are fitted to a straight line, this line may be estimated at some distance into the roadside.

The right road edge is in Fig. 8.6(a) often detected as traversable with only a few measurements marked as nontraversable. The traversable segments in the corridor are mostly terminated at the road edge, but many (about 40%) extend into the right roadside in this area.

In this area the vision-based road outline is superior to the laser scanner based corridor.

Many of the roadside areas along the gravelled roads suffer from this effect resulting in a low quality of the corresponding road edge line. Compare in Fig. 8.2 the left and right road edge line qualities at the shown situation –

about 120 m – with the road edge line quality of the gravelled road area from about 400 m and onwards.

8.7 Side roads

Detection of presence of side roads while passing may be a useful functionality for confirmation of own position or when searching for unexplored route alternatives. A situation with a side road is shown in Fig. 8.7.

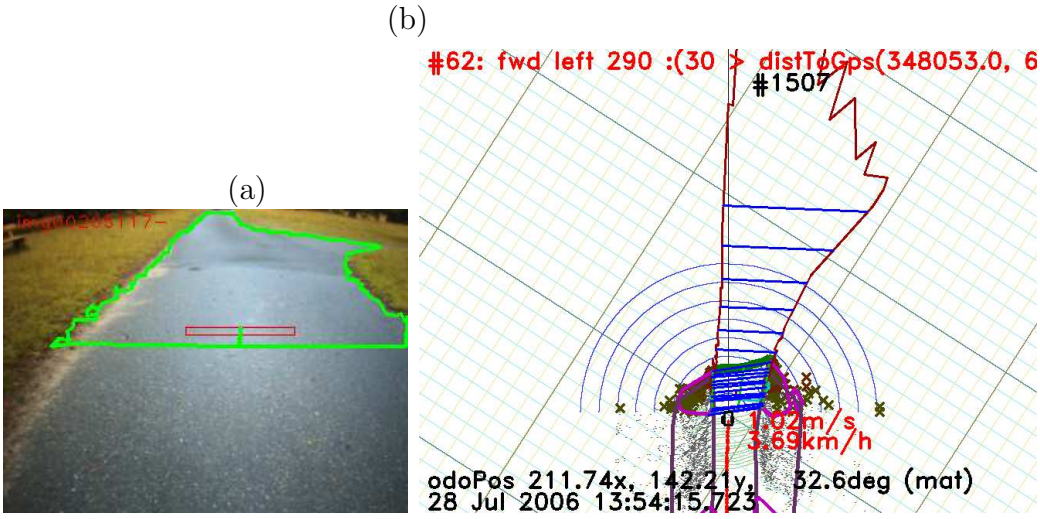


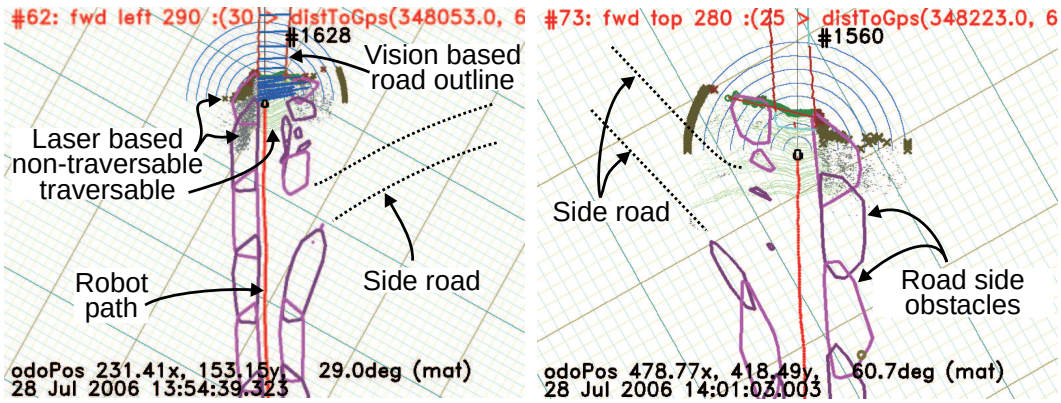
Fig. 8.7: The side road to the right is visible in the road outline, as seen from the robot camera in (a) and projected into planar view in (b).

The road outline shows a widening of the road, but this can be taken as a weak evidence only of the presence of a side road, the vision-based road outline may in other situations falsely detect a wider road – eg as in Fig. 4.7(h), where the right road edge is seen as road. When the side road gets closer it will be out of sight of the camera.

The laser scanner will detect parts of the side road while passing as shown in Fig. 8.8(a). The side road is visible as an opening in the line of obstacles, wide enough for the robot to pass; in this case an opening of about 5 m is found.

When going back in history to explore yet undiscovered territory such an opening would be adequate. To make a flag available for the navigation scheduler may be possible, but this could be sensitive to false detections during robot manoeuvring.

The road edge detection is a part of the laser scanner road edge detection and the combination of distance to the edge and the edge line quality may be usable as side road detection. The road lines for this side road are shown in Fig. 3.11(a).



(a) (b)

Fig. 8.8: The side road in (a) to the right as detected by the laser scanner. This leaves an open area to the right with no obstacles. The side road to the left in (b) is equally well detected.

The side road to the right, marked (5) in Fig. 8.1, is detected by the laser scanner as shown in Fig. 8.8(b). The edge lines for this side road are shown in Fig. 3.11(b).

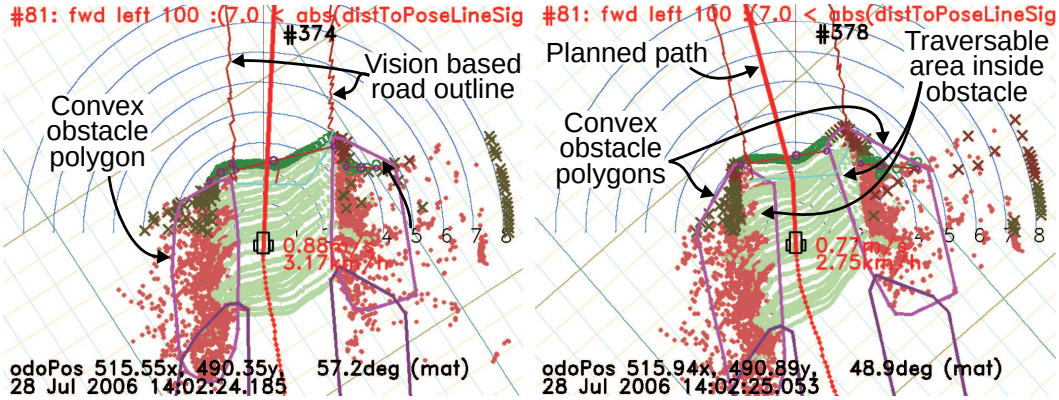
These two side roads are detectable from the data, but this is not always the case. At Fig. 8.1 the side roads just before and after the position marked (4) are not detected. The one to the left is no longer in use and is overgrown with grass. The other to the right is in use, but the surface of the side road is gravel whereas the main road is surfaced with asphalt. The result is that the road edge line follows the change in surface type as does the vision-based road outline.

8.8 Convex obstacles

Obstacles are stored as convex polygon areas with a size limitation. The assumption is that the error introduced using a convex polygon description is small compared to the calculation benefits obtained. Figure 8.9 shows a situation where parts of the area classified as traversable are included in an obstacle polygon.

In the situation in Fig. 8.9(a) the robot follows a rough left edge at some distance (1 m) and the obstacles generated cover the rough area on both sides of the road leaving about 3 m obstacle free road width; this is close to the optimal situation.

In Fig. 8.9(b) – four scans later – parts of the road have been classified as nontraversable. The obstacles on both sides are expanded to include the newly detected nontraversable areas. The expansion makes the convex-shaped



(a) (b)

Fig. 8.9: The situation in (a) produces obstacles to the left and to the right that cover the nontraversable area reasonably well. The situation in (b) is taken four scans later where part of the road is classified as nontraversable and the obstacles are expanded to include the newly detected obstacles.

obstacles include parts of the road, previously classified as traversable. The robot could potentially have been partially within one of these areas, which in turn could trigger unnecessary manoeuvres.

This situation is not critical in this case, but could be if narrower and more curved roads were to be passed. The solution could be to make the obstacle polygons smaller or to allow concave polygons, at the expense of the computational complexity.

8.9 Road ridge

At roads with a very high profile, where the laser scanner cannot see the other side of the road, the results may be undesired as shown in Fig. 8.10.

The robot is entering from a side road on to a crossing gravelled road (at (6) in Fig. 8.1). The crossing road has a higher level than the side road, and a combination of the road profile and the difference in level make the right side of the road surface beyond the horizon when seen from the laser scanner. At the 'ridge' the visible road edge may produce a road edge line with sufficient high quality to be used as an obstacle as shown in the figure.

At about this time the robot is supposed to cross the road to enter the side road on the other side. The road edge obstacles make the transition more difficult than intended (and in this case the robot had great difficulties in getting across).

The situation could question the benefit of using road side lines as obstacles, or indicate that the laser scanner height is insufficient for this type of road

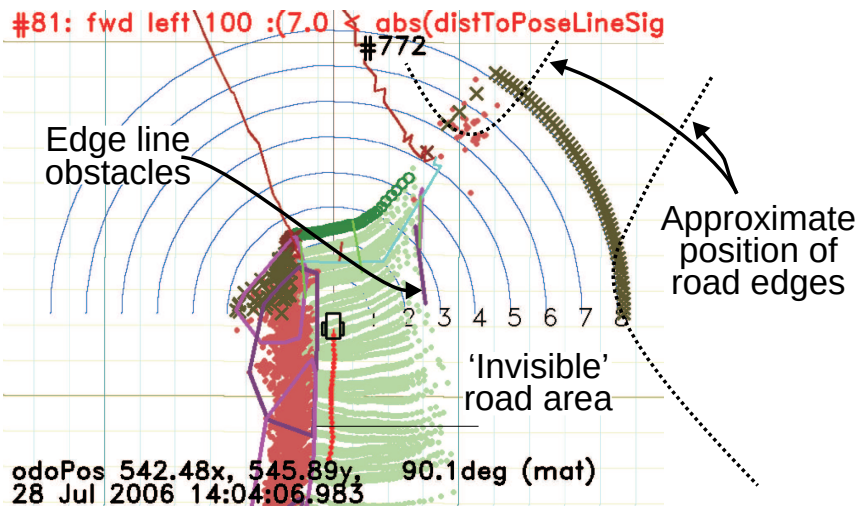


Fig. 8.10: The robot is entering from a side road to a higher crossing road. While climbing to the crossing road the full extent of the road is not visible to the laser scanner, and might at time produce road edge obstacles at the 'ridge'.

crossing (or both).

8.10 Odometry navigation

The open area in front of the hunting lodge is crossed using odometry based navigation. The area has a gravel surface and some accumulating odometry errors are expected. Figure 8.11 shows the odometry based robot path overlaid a satellite image of the area.

The overlaid image is aligned with the entry road and the distance from the entry road to the exit road is approximately 50 m. The misalignment of the detection of the exit road is small and presents no problem for the navigation script.

8.11 Discussion

The results are primarily obtained within the selected test area, but the desired abilities of the robot should be to be able to cope with more general real world situations. The focus on the test area is therefore to be seen as a subset of the desired navigation abilities. The ability to navigate within the test area should therefore not prohibit the ability to cope with other types of outdoor environments. The focus has thus been on the general abilities – detection traversable areas – and the ability to recognise topological features to assist navigation, eg along roads and in junctions.

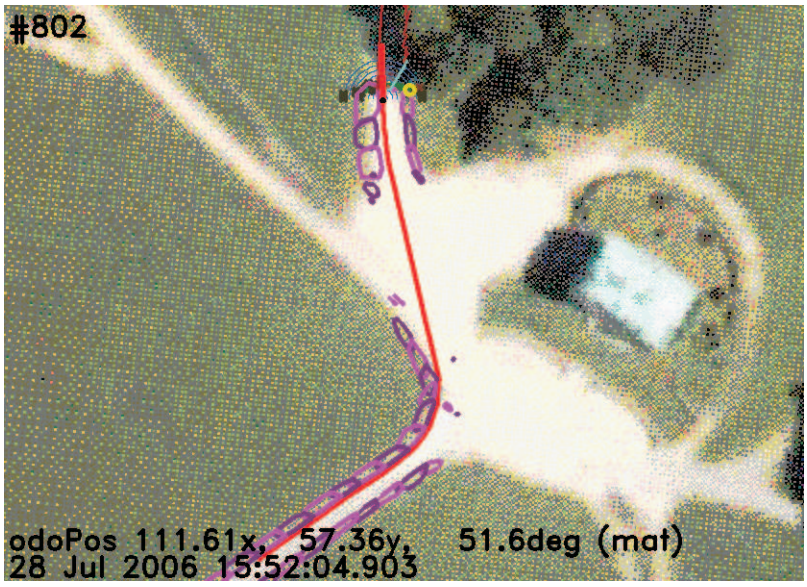


Fig. 8.11: Crossing of the hunting lodge area. The satellite image is overlaid with the odometry path of the robot. The distance across the open area is approximately 50 m.

The detection of road lines is valuable as it allows the robot to follow one of these lines. Most of the route is traversed following a road line. At the asphalt road leading to the hunting lodge (about 1.8 km) the left edge is followed all the way. The quality of the detection of the road lines is dependent on the distance to the line. Where the road profile has a clear peak at the centre of the road – and this is especially the case for the gravelled roads – the centre road line is easy to follow, and on most of the gravelled roads the robot follows this line.

The road lines are implemented as a calculation based on the most recent set of traversable segments; but it is not a continuous calculation. It is expected that a continuous calculation of the road lines – eg using a Kalman filter – could produce superior results in most situations.

The used robot is not equipped with tilt and roll sensors, and this makes it difficult to interpret measurements in situations with excessive tilt or roll. On this test route – while traversing pits or climbing to a higher level – a few situations produced an undesired behaviour in some of the tests. An alternative to a tilt sensor could be a closed loop servo that adapts the laser scanner tilt to a desired road detection distance; such a solution is expected to be able to solve some of the tilt related issues.

In open areas the navigation relies primarily on the ability to navigate on odometry. The only open area on the test route requires odometry navigation over a distance of about 50 m to hit an about 5 m wide exit road; this is not

problematic for the used robot.

The ability to classify the traversable road is adequate for the creation of traversable corridors that keep the robot on the road in almost all cases. In the few cases where the laser scanner based data would allow the robot to enter the roadside the vision-based road outline will limit the available route to the road.

The obstacles produced from the nontraversable measurements of the laser scanner mark the road edges and other obstacles satisfactory.

In some situations the produced obstacles are too restrictive. This is the case where road edge lines with high quality are used to form obstacles. The use of road edge lines as obstacles seem to create more problems than they solve. At the situations when they represent true road lines the obstacles are generated adequately without the use of road lines.

Chapter 9

Conclusion

Navigation of an autonomous robot is concerned with the ability of the robot to direct itself from the current position to a desired destination.

The thesis has successfully demonstrated a set of functionalities for mobile robot navigation. The combined system is primarily tested in an outdoor test area with a mixture of different road types and junctions. The robot has autonomously completed a 3 km test route in this environment.

The research has been concentrated on three areas: perception of the environment, transformation of the perceived information into behaviour and construction of a modular architecture.

9.1 Perception

Two main sensor types have been investigated for this environment: a laser scanner based road classifier and a vision-based road outline detector. The combined perception of these sensors has been demonstrated to be sufficient to direct the robot along the roads and across the junction in the test area.

9.1.1 Laser scanner perception

The used laser scanner is oriented towards the road at an angle of 9° from horizontal. This orientation is found to be about optimal for simultaneous detection of road and obstacles. The road is detected at a distance of about 2.6 m in front of the robot.

Road corridor

The measured range data from the laser scanner is classified into traversable and nontraversable measurements using a set of seven features extracted from

the data: raw height, roughness, step size, curvature, slope, width, and invalid data.

The traversable measurements of one laser scan are further grouped into traversable segments. Traversable segments from a number of scans are used to generate traversable corridors. The corridors describe traversable areas in which the robot should be able to navigate. The generated corridors are made available for the behaviour generation.

More than one corridor may be generated, eg: a corridor following the left and right side of an obstacle, road alternatives exist or one corridor for a cycle path and one for the main road. Sometimes a traversable part of the roadside generates an alternative corridor. Which corridor to follow is determined by the behaviour generation.

The corridor is used to extract further information from the traversed road.

Road lines

Road edge lines are estimated for the left and right limitations of the traversed road. A further road line is estimated for the road centre representing the highest point of the road. The road lines are used as reference lines in the behaviour generation.

The road lines include an estimation quality, and when this quality is high and the road edge is inside the laser scanner range, the road line represents accurately the true road edge.

Road type

The road type is estimated based on the corridor roughness. The road type is evaluated into the two types found in the test area: asphalt and gravel. The estimated road type is used to increase the edge detection sensitivity on smooth asphalt roads and is made available for the behaviour generation.

In most cases the road type is estimated correctly, but in a few cases a smooth gravelled road is estimated as asphalt. During the performed tests the road type has not been used to determine transition from one road segment to the next in the navigation script.

Obstacles

The nontraversable measurements are filtered into areas to be treated as obstacles. These obstacle areas are combined into convex shaped polygon areas. These areas represent roadsides which are nontraversable as well as other obstacle types – like pedestrians. The obstacles are made available for the behaviour generation.

The obstacles generated have reliably been used to avoid stationary obstacles.

9.1.2 Vision-based perception

The robot camera is viewing the area in front of the robot – from about 1 m to the horizon. Two data types are extracted from the camera images: road outline and artificial guidemark positions.

Road outline

The camera image is segmented into road and not road based on a seed area that is known to be road. A balanced combination of chromaticity and edge detection is used to determine the transition from road to non-road. The result is a road outline polygon encapsulating the seed area. The image chromaticity is compensated (based on pixel intensity) to reduce the effect of shadows. The road outline is made available for the behaviour generation.

In most cases the road outline represents the true road limitations, but is less reliable in a number of situations: when the seed area partially includes a roadside area and when the colour of obstacles or roadsides is too close to the road colour. Hard shadows and road paintings are further often estimated as obstacles.

The estimated road outline is able to detect an obstacle that crosses the road outline only. Obstacles encircled by the road outline are not detected and typically remain undetected until inside coverage of the laser scanner.

The road outline is projected to robot coordinates under the assumption that the road is in the plane of the robot base, when this assumption is violated, eg during robot roll and tilt, the accuracy of the road outline is reduced. The road outline covers the area beyond the laser scanner range.

Artificial guidemark

A guidemark sensor function is investigated; this function allows identification of guidemarks and estimates the position and orientation of the guidemark in 3D.

The guidemark is a flat checkerboard type frame with a coded identification pattern at the centre. The frame is square and is well suited for position estimation. The code in the centre can be used to make uniquely identifiable guidemarks.

The frame consists of alternating black and white squares. Each square must be covered by at least about 4 pixels in the camera image to allow detection and position estimation. The 3D position and orientation of the guidemark can

be estimated with an absolute accuracy better than 5 cm and 5°, respectively (at the conditions described in chapter 4).

The guidemark ID and the 3D guidemark posture is made available for behaviour generation.

9.2 Behaviour generation

The behaviour generation utilises the perceived sensor information to avoid obstacles and to make progress in the mission.

The cognitive skills are limited to the utilisation of the perceived data for obstacle avoidance. The remaining high level cognitive skills – determining how the robot should position itself on the road and how to cross junctions – are left to a navigation script.

Navigation script

The navigation script describes how the robot is to behave when following a road, how to detect junctions and how to cross the junction to the next road. The navigation script interpreter has a number of drive modes available, each drive mode has a number of options affecting the behaviour. The drive modes include: following a road edge line at a given distance and aiming for a position (in robot or odometry coordinates).

The script further allows calculation and branching based on the available data. The available data includes – in addition to the perceived sensor information – the historic posture and the robot status. A watch function is available, and this could – as an example – be used to monitor the current heading to detect when the robot has entered into a side road or a junction.

Sensor fusion

The vision-based road outline is fused with the laser scanner based corridors to a corridor based on both sensors.

If the vision-based road outline fails for some reason, the navigation continues based on the laser scanner data only.

The combined corridor allows steady road following, both in the situations where the laser scanner has difficulties in separation of the road from the roadside, and in situations where the road outline fails (when both fails simultaneously the performance is degraded). The vision-based road outline allows the robot to start an obstacle avoidance manoeuvre at an early stage. When a person is avoided in this way, the intensions of the robot will be visible sufficiently early to limit potential conflicts.

Obstacle avoidance

The obstacle avoidance behaviour is the planning of a short-term route from current position to a destination position – also called the exit posture – inside the coverage of the sensors.

The exit posture is typically selected inside an available corridor, and the exit posture is selected for the best possible fulfilment of the current objectives in the navigation script.

A route is planned from the current posture to the exit posture using a visible graph method. The method starts by testing the direct route from current position to the exit position, if an obstacle is too close, an avoidance route is created, as well as the most likely alternatives. The curvature of the route is limited to impose restrictions on the dynamic behaviour of the robot. The path is constructed using a combination of straight and curved elements, where the turning radius in the curved part is selected to limit the centripetal acceleration.

More than one obstacle avoidance route may be possible and the most optimal of the available routes are selected based on a set of criteria including: route roughness, required manoeuvres and distance.

The path calculation method is fast when there is few obstacles only, but the processing time increases exponentially with the number of obstacles found on the path. In the test area the obstacle count is small and thus no real challenge to the selected method.

Moving obstacles are not considered.

9.3 Architecture

A new software architecture is proposed as the available open source architectures were found to be too restrictive.

The proposed architecture is selected to allow the functionality to be implemented in components. The components communicate in a client server architecture using XML formatted messages. Each of the components further allows plugins to facilitate enhanced or supplementary functionality without replacement of a full component.

Three components are built using this architecture: a laser scanner server, a camera server and a behaviour server.

The laser scanner and vision sensors have high bandwidth requirements. This high bandwidth data is usually reduced significantly during the feature extraction from the sensor data. This sensor processing is therefore best performed as a plugin function in the server component for the sensor. The perceived data can then be transferred to other components as needed.

The plugin and the component structures encourage a division of complex functionality into manageable sizes. The system knowledge needed to be able to create a plugin or use a server component is small, and thus new functionalities can be coded and tested within a short time frame. This is especially significant in a university environment, where students at a single course can create and test advanced functionalities. The architecture is believed to be advantageous where distributed development and maintenance are essential.

The maintenance of the components and – to some extent – the plugins are independent. The use of XML formatted communication enables extension of the data structure from one server without the need for clients to be updated concurrently (if they do not need the extended information).

The architecture is not fully developed, but the camera server and the laser scanner server have already proved their value in a number of projects.

9.4 Results

Test results have been obtained over a period of almost one year and in a number of weather conditions including sunshine, overcast and wet weather, but excluding snow and heavy rain.

The road following is mostly unproblematic; especially asphalt roads where the roughness difference between the road surface and the roadsides is constantly high. On gravelled roads the difference in roughness of the road and parts of the roadside is sometimes small. The corridor edges are here more ragged than on asphalt, the consequence is a lower estimation quality for the left and right road lines. The road centre line is usually estimated with high quality on the gravelled roads and is easy to follow.

Fork junctions are mostly passed by following the left or right road lines into the desired road. More complicated junctions are detected when entering into the junction and from here the behaviour script directs the robot towards the desired exit road.

The navigation script is manually prepared using a map of the area and subsequently adjusted after tests in the area. It has not been possible to complete an unrehearsed test mission successfully; especially the crossing of junctions needs script adjustments.

There are season and weather variations in the performance of the robot. In sunshine the hard shadows from obstacles (eg pedestrians) are taken as obstacles by the vision-based road outline sensor. The slightly more scattered shadows from trees are less of a problem. During the winter and early spring the vegetation on the roadsides may be more flat and having a more greyish colour, this is slightly more problematic for both the laser scanner and vision-based sensors – resulting in less smooth road following. Wet weather and partially

wet roads are handled reasonably well. Snow and heavy rain conditions are not tested, and the sensors are not expected to produce usable results in these cases.

The required update rates for laser scanner, vision and obstacle avoidance are speed dependent and is selected to be as low as possible. The laser scanner update rate is about 6 Hz at a robot speed of $1\,\mathrm{ms}^{-1}$ to allow detection of obstacles larger than 5 cm. The vision-based road outline is detected at a slower 1 Hz rate, as this is the long range road detection (from about 2.6 m). The obstacle avoidance path planning was running at a 2 Hz rate, as this should be sufficient for obstacle avoidance manoeuvring in the available speed range (up to $1.5\,\mathrm{ms}^{-1}$).

9.5 Future work

For the presented solutions there are a number of enhancements that is expected to improve the obtained results. These include the following subjects.

- Improvements of the calculation of road lines. The calculation could probably be improved by using a continuous calculation, eg a Kalman filter. The road lines should neither use measurements close to the laser scanner maximum range nor measurements close to a visible horizon.

- A permanent watch function should be included to detect if the robot has entered a side road or the road has made a sharp turn. Such a function could be based on deviation from a previously stable course. This is the used method to detect junctions and would simplify the navigation script.

- The vision-based road outline could be searched for encircled obstacles. This is a rather simple enhancement and would remove an undesired feature from the presented solution, where obstacles with a height up to the height of the camera may remain undetected.

- A roll and tilt sensor could improve the coordinate conversion of measurements, resulting in improved performance in a number of situations.

- An improved GPS handling could allow a drive mode aiming at a GPS position directly. Such a solution would not improve the results directly, but could potentially simplify the navigation script in some situations.

- In the implemented software architecture the resource-providing plug-ins do not use a standardised method. Such a standardised method for resource sharing should be provided to facilitate a more decentralised maintenance structure.

For the long perspective of navigation autonomy, the following list of subjets shows the areas, which would have been investigated had time been available:

- Vision-based perception with classification of the seen objects in classes like: trees, signposts, cars, pedestrians and buildings. This is envisaged as a combination of image segmentation and 3D reconstruction using eg the 'structure from motion' method. This is a complicated perception sensor, of wich the road outline sensor described in this thesis is seen as a small initial step.

- A mapping function where the road outline and other identified objects create and update a map while moving. The map elements are assumed to be positioned primarily based on GPS positions, with an allowance for correction of errors using one of the methods seen in the SLAM community, eg the 'consistent pose estimation' method described in Konolige (2005) for example or Folkesson et al. (2005).

- With a map in place with some experienced navigation information, the cognitive abilities may be enhanced to increase the autonomy, eg while following roads or crossing junctions, where experience could determine the drive mode rather than an operator produced script.

The research is not expected to lead to autonomous robots capable of coping with public traffic during the rush hour initially. The robot autonomy will probably develop in isolated areas where the rules of behaviour are less demanding, like in private houses, on agricultural farmland or on the military battlefields.

Bibliography

Albus, J. S. (1992), A reference model architecture for intelligent systems design, *in* P. J. Antsaklis & K. M. Passino, eds, 'An Introduction to Intelligent and Autonomous Control', Kluwer Academic Publishers, Boston, MA, pp. 57–64.

Albus, J. S., Lumia, R. & McCain, H. G. (1986), Nasa/nbs standard reference model for telerobot control systems architecture, Technical Report NBS Tecnical note #1235 or NASA document SS-GSFC-0027, National Bureau of Standards (USA).

Andersen, J. C., Andersen, N. & Ravn, O. (2004), Trinocular stereo vision for intelligent robot navigation, *in* '5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles', Lisboa, Portugal.

Andersen, J. C., Blas, M. R., Andersen, N., Ravn, O. & Blanke, M. (2006), 'Traversable terrain classification for outdoor autonomous robots using single 2D laser scans', *Integrated Computer-Aided Engineering*.

Aufrere, R., Gowdy, J., Mertz, C., Thorpe, C., Wang, C.-C. & Yata, T. (2003), 'Perception for collision avoidance and autonomous driving', *Mechatronics* **Vol. 13 Issue. 10**, 1149–1161.

Babuška, R. (1998), *Fuzzy Modeling for Control*, Kluwer Academic Publishers.

Barontini, G., Segurini, G., Breemen, A. v., Brodtman, T., Buckingham, R., Christensen, H. I., Ikonomopoulos, S., Gelin, R., Fournier, R., Gu, C.-Y., Guettier, C., Hgele, M., Isasi, L., Koeppe, R., Bischoff, R., Patin, B. & Pegman, G. (2005), 'Building europ, the european robotics platform, the high level view', http://www.robotics-platform.eu.com.

Behringer, R., Travis, W., Daily, R., Bevly, D., Kubinger, W., Herzner, W. & Fehlberg, V. (2005), Rascal - an autonomous ground vehicle for desert driving in the darpa grand challenge 2005, *in* 'Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE', IEEE, pp. 644–649.

Bertozzi, M. & Broggi, A. (1997), 'Vision-based vehicle guidance', *Computer* **Vol. 30 Issue. 7**, 49–55.

Blanke, M., Kinnaert, M., Lunze, J. & Staroswiecki, M. (2003), *Diagnosis and Fault-tolerant Control*, Springer Verlag.

Blas, M. R., Riisgaard, S., Ravn, O., Andersen, N., Blanke, M. & Andersen, J. C. (2005), Terrain classification for outdoor autonomous robots using 2D laser scans., *in* '2nd int. Conf. on Informatics in Control, Automation and Robotics, ICINCO-2005.', Barcelona, 14-17 September, pp. 347–351.

Brooks, A., Kaupp, T., Makarenko, A., Williams, S. & Oreback, A. (2005), Towards component-based robotics, *in* 'Intelligent Robots and Systems, 2005. (IROS 2005).', IEEE/RSJ, pp. 163–168.

Brooks, R. A., ed. (1990), *The Behavior Language; User's Guide*, MIT Press.

Carstensen, J. M., ed. (2001), *Image analysis, vision, and computer graphics*, Informatics and Mathematical Modelling, Technical University of Denmark, DTU.

Douglas, D. & Peucker, T. (1973), 'Algorithms for the reduction of the number of points required to represent a digitized line or its caricature', *Canadian Cartographer* pp. 112–122.

Fergus, R., Perona, P. & Zisserman, A. (2003), Object class recognition by unsupervised scale-invariant learning, *in* 'Proceedings. 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition', IEEE Comput. Soc, pp. II–264.

Folkesson, J., Jensfelt, P. & Christensen, H. (2005), Graphical slam using vision and the measurement subspace, *in* 'Proceedings of International Conference on Intelligent Robots and Systems 2005', IEEE, pp. 325–330.

Guivant, J. E., Masson, F. R. & Nebot, E. M. (2001), 'Optimization of the simultaneous localization and map-building algorithm for real-time implementation', *IEEE Transaction on Robotics and Automation* **17**(3), 242–257. The University of Sidney, Australia.

Henriksen, L. & Krotkov, E. (1997), Natural terrain hazard detection with a laser rangefinder, *in* 'IEEE International Conference on Robotics and Automation', Vol. 2, pp. 968–973.

Horswill, I. (1994), Visual collision avoidance by segmentation, *in* 'Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems '94. 'Advanced Robotic Systems and the Real World', IROS '94.', IEEE, pp. 902–909.

Jochem, T., Pomereau, D. & Thorpe, C. (1993), Maniac: a next generation neurally based autonomous road follower, *in* F. Groen, S. Hirose & C. Thorpe, eds, 'Intelligent Autonomous Systems. IAS-3. Proceedings of the International Conference', IOS Press, pp. 592–601.

Jochem, T., Pomerleau, D. & Thorpe, C. (1995), Vision-based neural network road and intersection detection and traversal, *in* 'Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots", IEEE Comput. Soc. Press, pp. 344–349 vol.3.

Kadir, T. & Brady, M. (2001), 'Saliency, scale and image description', *International Journal of Computer Vision* **Vol. 45 Issue. 2**, 83–105.

Kim, D. & Nevatia, R. (1999), 'Symbolic navigation with a generic map', *Autonomous Robota* **Vol. 6**, 69–88. Soongsil University, Korea and University of Southern California, USA.

Klöör, P. L., Lundquist, P., Ohlsson, P., Nygårds, J. & Wernersson, A. (1993), Change detection in natural scenes using laser range measurements from a mobile robot, *in* 'Proceedings of 1st IFAC International Workshop on Intelligent Autonomous Vehicles', IFAC, University of Southampton, pp. 71–76.

Konolige, K. (2005), Slam via variable reduction from constraint maps, *in* 'Proceedings of the International Conference on Robotics and Automation, 2005', IEEE, pp. 667–672.

Kragic, D.; Christensen, H. (2005), 'Advances in robot vision', *Robotics and Autonomous Systems* **Vol. 52 Issue. 1**, 1–3.

Liatsis, P., Goulermas, J. & Katsande, P. (2003), A novel lane support framework for vision-based vehicle guidance, *in* 'Proceedings of the Industrial Technology, 2003 IEEE International Conference', IEEE, pp. 936–941 Vol.2.

Loaïza, H., Triboulet, J., Lelandais, S. & Barat, C. (2001), 'Matching segments in stereoscopic vision', *IEEE Instrumentation and Measurement Magazine* pp. 37–42.

Loaïza, H., Triboulet, J., Lelandais, S., Chavand, F. & Artigue, F. (1999), A multi-configuration stereoscopic vision system for domistic mobile robot localization, *in* 'Robot Motion and Control, 1999. RoMoCo '99. Proceedings of the First Workshop on', IEEE, pp. 207–212.

Macedo, J., Matthies, L. & Manduchi, R. (2000), Ladar-based discrimination of grass from obstacles for autonomous navigation, *in* 'Experimental Robotics VII, Proceedings ISER 2000, Waikiki, Hawaii', Springer, pp. 111–120.

Mejnertsen, A. & Reske-Nielsen, A. (2006), Control of autonomous tractor, Master's thesis, Ørsted•DTU, Technical University of Denmark, DK-2800 Kgs. Lyngby.

Mizoguchi, F., Nishiyama, H., Ohwada, H. & Hiraishi, H. (1999), 'Smart office robot collaboration based on multi-agent programming', *Artificial Intelligence* **Vol. 114 Issue. 1–2**, 57–94.

Montemerlo, M. & Thrun, S. (2004), A multi-resolution pyramid for outdoor robot terrain perception, *in* 'Proceedings - Nineteenth National Conference on Artificial Intelligence (AAAI-04)', compendex, pp. 464–469.

Montemerlo, M., Roy, N. & Thrun, S. (2003), Perspectives on standardization in mobile robot programming: the carnegie mellon navigation (carmen) toolkit, *in* 'Proceedings. 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003)', IEEE, pp. 2436–2441 vol.3.

Murray, D. & Little, J. J. (2000), Using real time stereo vision for mobile robot navigation, *in* 'Autonomous Robots', Kluwer Academic Publishers, The Netherlands, University of British Columbia, Canada, pp. 161–171.

Nielsen, A. & Breiting, M. (2004), Design and implementation of outdoor mobile robot, Master's thesis, Technical University of Denmark, Ørsted•DTU, Automation.

Ponweiser, W., Vincze, M. & Zillich, M. (2005), 'A software framework to integrate vision and reasoning components for cognitive vision systems', *Robotics and Autonomous Systems* **Vol. 52 Issue. 1**, 101–114.

Riisgaard, S. & Blas, M. R. (2005), Navigation for outdoor mobile robot, Master's thesis, Ørsted•DTU, Technical University of Denmark, DK-2800 Kgs. Lyngby.

Robowatch (2006), 'Ofro surveillance robot by robowatch technologies gmbh', http://www.robowatch.de.

Se, S., Lowe, D. & Little, J. (2001), Vision-based mobile robot localization and mapping using scale-invariant features, *in* 'IEEE International Conference on Robotics and Automation, Seoul, Korea', IEEE, pp. 2051–2058.

Shah, S. & Aggarwal, J. K. (1997), 'Mobile robot navigation and scene modelling using stereo fish-eye lens system', *Machine Vision and Application* **10**, 159–173.

Siegwart, R. & Nourbakhsh, I. R., eds (2004), *Introduction to Autonomous Mobile Robots*, MIT Press.

Simmons, R. (2005), 'Inter process communication (ipc)', http://www.cs.cmu.edu/afs/cs/project/TCA/www/ipc/index.html.

Simmons, R. & Apfelbaum, D. (1998), A task description language for robot control, *in* 'Proceedings Conference on Intelligent Robotics and Systems'.

Thorpe, C., Jochem, T. & Pomerleau, D. (1997), The 1997 automated highway free agent demonstration, *in* 'Proceedings of Conference on Intelligent Transportation Systems, 1997. ITSC '97.', IEEE, pp. 496–501.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V. & Pascal, S. (2006), 'Stanley: The robot that won the darpa grand challenge', http://cs.stanford.edu/group/roadrunner/.

Trepagnier, P. G., Kinney, P. M., Nagel, J. E., Doner, M. T. & Pearce, J. S. (2005), Team gray techical paper, Technical report, Gray & Company Inc.

Urmson, C., Anhalt, J., Clark, M., Galatali, T., Gonzalez, J. P., J., G., Gutierrez, A., Harbaugh, S., Johnson-Roberson, M., Kato, H., Koon, P. L., Peterson, K., Smith, B. K., Spiker, S., Tryzelaar, E. & Whittaker, W. R. L. (2004), High speed navigation of unrehearsed terrain: Red team technology for grand challenge 2004, Technical Report CMU-RI-TR-04-37, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Vandapel, N., Huber, D., Kapuria, A. & Hebert, M. (2004), Natural terrain classification using 3-d ladar data, *in* 'Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on', IEEE, pp. 5117–5122.

Vaughan, R., Gerkey, B. & Howard, A. (27-31 Oct. 2003), On device abstractions for portable, reusable robot code, *in* 'Proceedings. 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003)', IEEE, pp. 2421–2427 vol.3.

Wallace, R., Matsuzaki, K., Goto, Y., Crisman, J., Webb, J. & Kanade, T. (1986), Progress in robot road-following, *in* 'Proceedings 1986 IEEE International Conference on Robotics and Automation (Cat. No.86CH2282-2)', IEEE Comput. Soc. Press, pp. 1615–21 vol 3.

Wijesoma, W., Kodagoda, K. & Balasuriya, A. (2004), 'Road-boundary detection and tracking using ladar sensing', *Robotics and Automation, IEEE Transactions on* pp. 456–464.

World Wide Web Consordium, . (2004), Extensible markup language (xml) 1.1, Technical Report http://www.w3.org/TR/xml11/, W3C.
World Robotics

World Robotics *(2005), United Nations Economic Commission for Europe (UNECE) and International Federation of Robotics (IFR).*

Xiang, Z. & Ozguner, U. (2005), Environmental perception and multi-sensor data fusion for off-road autonomous vehicles, in *'Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE', IEEE, pp. 584–589.*

Appendix A

Trinocular stereovision

This article was presented at 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles (Andersen et al. (2004)).

Abstract

This paper describes a vision sensor that extracts visible features of objects using a set of on-board robot cameras. The purpose of the sensor is to be able to classify the seen objects into abstract object types like tables, chairs, walls and doors using these features. The results show that the vision sensor is able to extract a filtered set of features suitable for the purpose. The method used is stereoscopic scanning followed by a filtering process in 3D space. The method is insensitive to the shape of the objects and the structure of the background.

A.1 Introduction

The research project behind this paper is focused on autonomous robot navigation in an indoor human environment, where the robot is able to find and recognize some basic types of objects. The robot sensors include camera vision as the primary sensor for this task.

Mobile robot navigation is the discipline of locating own position and to find a path to the destination. Finding a path involves in this case recognition and classification of observed objects, so that the destination can be described in terms of the classified objects, e.g. as "go to the third table in the first room to the left".

The stereoscopic vision sensor provides the majority of the data needed for object classification. The extracted data should allow for classification into classes like chair, table, wall, door or human, without prior knowledge of where to expect what type of objects.

Stereoscopic vision systems have been described in a number of papers over the resent years. Most of these systems extract linear features in the images from single cameras and then correlate these to get the 3D information like in Loaïza et al. (2001), Loaïza et al. (1999), Kim & Nevatia (1999) and Shah & Aggarwal (1997). In Se et al. (2001), the images are searched for scale invariant features — e.g. line ends — before correlation and 3D calculation.

Most of the stereoscopic camera solutions use three cameras achieve better immunity to false correlations. Image processing for stereo calculation is processing power intensive and only a few papers document image processing times of less than one second. For obstacle mapping Murray & Little (2000) expect 3 updates per second, and, for robot position estimating Se et al. (2001), two estimates per second were obtained from the camera images using a 700 MHz PC. Faster is better, but processing-times in excess of one second is acceptable when the resulting data is to be used for route planning and not directly in the motion control loop.

This paper describes a method where images are scanned for 3D points first and the filtering is done subsequently in 3D space.

A.2 Objectives and Overview

When looking at an image as in Figure A.1 it is easy to see that it shows a chair, a golf-ball on the floor, and some closets in the background. If a robot is to get to the same conclusion, there is a fair bit of processing is to be performed, some of which is described in the following.

To find out that the nearest object is a chair, a number of tasks must be performed. The object must be separated from the rest of the image, and it must be analyzed for properties that indicate it is a chair. The properties could be that it has a surface, the seat, of an appropriate size (e.g. 40x40 cm) at an appropriate height (approximately 50 cm), and that it has an open structure below the surface to support it. These properties would make it distinguishable from other objects like a box or a table.

From one images set the visible features can be extracted. The visible features of a chair could be the front edge of the seat, the chair bag or parts of the support structure. The idea is that a set of such features is sufficient to perform the needed object classification.

This paper describes a vision sensor and the processing behind the feature extraction.

The sensor has three major parts, the feature filtering, the stereo scanning and the camera calibration. Each of these parts are described below.

Fig. A.1: An image like this is easily decoded by a human, but requires explicit processing for a robot.

A.3 Feature filtering

The data from the stereo correlation is 3D positions reconstructed from corresponding points in a set of stereoscopic images. About 1000 3D positions are produced in a full stereoscopic image scanning. These positions are grouped into spatially separated clusters.

The feature filtering function must ensure that clusters represent one object only. As different objects often is seen close together — e.g. a table and a wall — they may not be separable on distance alone.

To separate these clusters into features from just one object, two primary methods are used. The first is to extract horizontal and vertically oriented features, and the second is to split combined features based on similar positional and color properties.

In an indoor environment, horizontal and vertical features are often found in especially empty rooms. These features are extracted first based a density histogram along the vertical axis for the horizontal lines. This will typically produce a peak in the histogram when the cluster contains a major feature

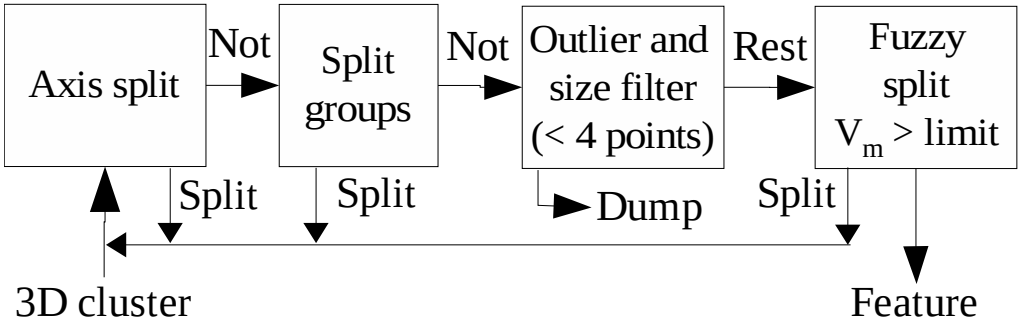


Fig. A.2: The input clusters are groups of isolated 3D detections, these gets divided into features of just one object by splitting.

perpendicular to the vertical axis. The peak is then isolated to a new cluster. The same method is used across the image, and this isolates primarily vertical lines.

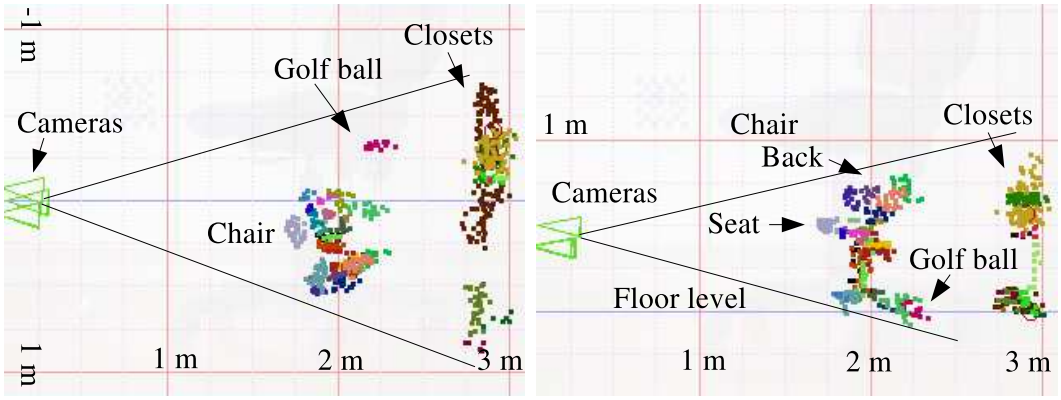


Fig. A.3: The images show the features extracted from the scene in Figure A.1. The left image is top view, and the right image is side view. The three camera positions are shown as triangles. Each feature is represented with a colored group of 3D detections.

A fuzzy c-means classifier from Babuška (1998) is able to separate 3D positions in a cluster of into two or more sub-clusters. The ellipsoid shape of the sub-clusters may differ, e.g. a cluster representing a chair seat in combination with one of its legs, may be divided into a narrow ellipsoid formed cluster from the leg and a more ball formed for the seat. The fuzzy c-means method needs to know the number of sub-clusters, and the computation afford increases rapidly with the number of clusters. A computational affordable method is to split clusters into two subclusters until the cluster statistics are reasonable, or the

clusters are too small to split. This is the method used, and it produces slightly more clusters than is strictly necessary, as the only drawback.

The cluster statistics is calculated based on the best-fit 3D line. The 3D line is the major axis of the ellipsoid that best describes the cluster, i.e. where the sum of the squared distance from the line to the 3D positions is minimum. The line is found by finding the best fit line in two dimensions twice. First in the horizontal plane and then in the 2D plane spanned by the vertical axis and one of the horizontal axes. The used axis combinations are selected so that singularities are best avoided.

A number of statistics are calculated along this 3D line with the endpoints $\mathbf{E}_1$ and $\mathbf{E}_2$. One of them is the mass distribution or inertial moment V_m for a unified line length, as shown in Equation A.1.

$$V_m = \frac{\frac{1}{n}\sum_{j=1}^{n}\big(t(\mathbf{X}_j) - t(\mathbf{X}_g)\big)^2}{|\mathbf{E}_2 - \mathbf{E}_1|^2} \tag{A.1}$$

Where $\mathbf{X}_g$ is the center of gravity and $t(\mathbf{X}_j)$ is the position on the line closest to the 3D point $\mathbf{X}_j$, all 3D points are given equal weight. For a uniform distribution along the 3D line, this value V_m would be $^1/_{12}$. Clusters are split if the mass distribution is above a value comparable with the uniform distribution.

Clusters are furthermore tested for outliers and for splits. Splits are clusters that are no longer continuous, i.e. there is an opening along the major cluster axis. Split clusters are divided into continuous subclusters.

The full feature extraction process is shown in Figure A.2, and an example of the extracted features is shown in Figure A.3.

The resulting clusters are assumed object features and are shown on top of one of the images in Figure A.4.

A.4 Stereoscopic scanning

The use of cameras to aid robot navigation has been described in a number of papers using a number of different approaches. A two camera solutions with fish-eye lenses is used for navigation in Shah & Aggarwal (1997) by finding significant lines ending in the geometric vanishing points, and from these and vertical lines build a model of the traversed corridor. Another project Loaïza et al. (1999) and Loaïza et al. (2001) uses two cameras mounted on top of each other and does stereoscopic calculation on contrast straight lines found in the images, the Loaïza et al. (2001) project puts emphasis in color on each side of the extracted lines to improve matching. The project Se et al. (2001) uses three cameras in a right angle triangular configuration to remove false correlations. The project also pre-extracts a number of (scale invariant) features before stereo

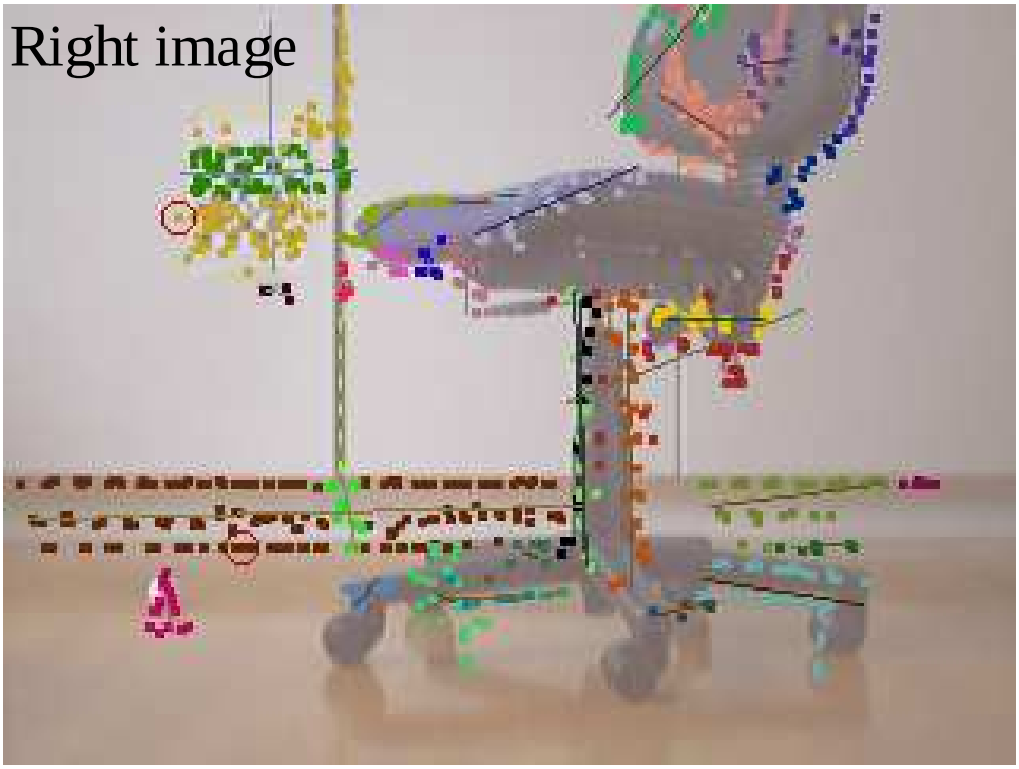


Fig. A.4: The resulting clusters after filtering is shown on top of one of the original images. In total 848 3D detections are shown in 36 clusters.

calculation, and then stores these points (about 3000 for one room) for the ongoing navigation.

These projects all put a great deal of effort into extracting data from the two-dimensional images prior to the stereo calculation. This will reduce the stereo calculation effort, but objects are limited to the extracted types, typically linear features only. The method selected here does not extract features in the two-dimensional images but only edges. This ensures that no object gets discarded based on its shape.

The camera configuration is as shown in Figure A.5. The cameras are standard web cameras connected to the robot motherboard using a Universal Serial Bus (USB) interface. The robot main processor is using a Linux operating system (Redhat standard distribution).

The stereo scanning is performed using two camera images at a time. The reference image is divided into a number of scanning lines and for each of these the corresponding epipolar line is found in the other image. Both images are then resampled along these lines in 3 pixel wide bands. Contrast edges are found in the resampled reference image, and for each of these, a match is found

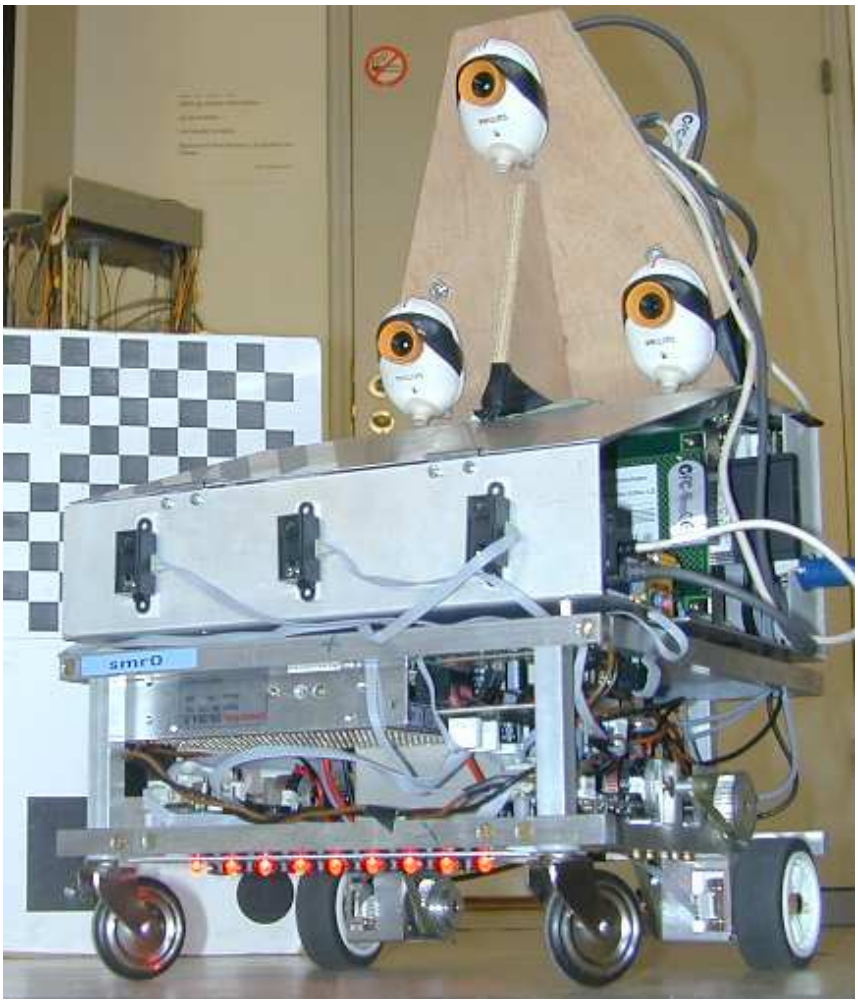


Fig. A.5: The robot platform used for the experiments with the trinocular camera mount on top. The distance between any two cameras are 15 cm.

in the other image. The cross-correlation coefficient ρ in range $[-1, 1]$ is found over a small mask area. The size of this correlation mask area depends of the resolution, and is 3x17 pixels at a 120x160 image resolution and 3x31 pixels for a 240x320 image resolution. The image is scanned in the part of the image that would result in a 3D distances from 0.5 meter to 20 meter from the cameras. To reduce processing the correlation is calculated for every pixel position is the last correlation was a close match — i.e. a ρ value above 0.2. For ρ below this value, from 1 to 6 pixels gets skipped before another correlation is attempted.

Figure A.6 shows an example on a top and bottom-right image set. One of the edges on the floor line in the left image is found in the search area in the right image. The correlation coefficient is shown in the box in the right image,

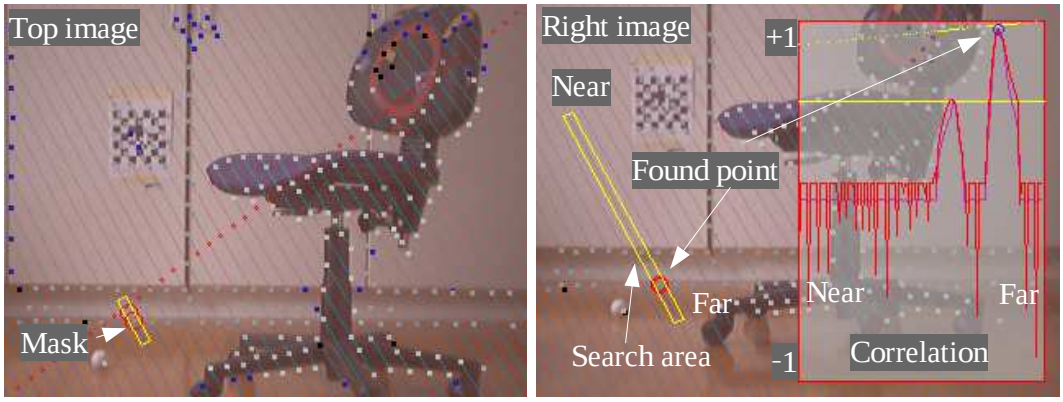


Fig. A.6: The left and right image shows the epipolar lines used in the scanning process. The left image is the reference image (top camera) with one edge marked in the mask-sized box. The area marked in the right image is scanned for the same pattern. The correlation quotient is shown in the curve. The gray dots show the points where correlation is attempted — darker gray is further away.

with the best correlation marked with a circle. The used image resolution is 320x240 pixels.

Each set of two images is scanned in this way, both with the first and the second image as the reference image. In total six scannings are thus performed for each set of three images.

The resultant set of 3D positions (about 1200 for the chair motif) are grouped into clusters, so that the 3D positions in a cluster are not separated by more than could be expected with the used epipolar line distance (9 pixels).

Each cluster should have 3D positions from at least four of the six scannings, otherwise the cluster is assumed false and dropped. This criterion removes almost all the false correlations. The remaining clusters are split and filtered as described above.

A.5 Camera calibration

The camera orientation is very important for a correct stereoscopic conversion. The used Web camera could not be mounted in a predetermined orientation with a reasonable accuracy, so the orientation must be determined after the cameras are mounted on the robot.

The general idea is that the camera *position* on the robot can be established reasonably accurate by measurement, whereas the *rotation* is difficult to establish by direct measurement. By placing a known calibration chart at

some distance directly in front of the robot at a known height, it is possible to calculate a sufficient exact rotation around all three axes for each camera on the camera mount.

The calibration chart consists of a number of black squares on a white background, arranged as a checkerboard as shown behind the robot in Figure A.5. The corner where two black squares touch is scale invariant and easily detected. Two of the black squares in the calibration chart are omitted and the position of these are used to determine the center of the calibration chart and from that the pixel position of all corners.

The pixel position $\mathbf{X}_p$ $[x_p, y_p]$ corresponding to each corner position $\mathbf{X}_c$ $[x_c, y_c, 0]$ on the flat calibration chart can be calculated as a function of chart position relative to the camera $[x_t, y_t, z_t]$ and the camera rotation $[\Omega, \Phi, \kappa]$ around the three axis. The pixel position $\mathbf{X}_p$ and the corresponding point $\mathbf{X}_c$ on the calibration chart are known for up to 100 positions. The camera position and rotation form this equation is then estimated using a least-square parameter adjustment method.

This method can determine the orientation of the cameras to about 0.01° for both the left-right orientation Φ and the up-down orientation Ω, and to about 0.03° for the rotation κ around the camera optical axis.

When using identical cameras with USB interfaces there is no guarantee that the cameras will be detected in the same order as last time they were plugged into the robot. The calibration method described can estimate the camera position to within 1–2 cm, which is sufficiently accurate to determine which camera device is at top, left and right position.

The used camera (Philips PCVC 740K) has a focus length of about 1055 pixels with an image resolution of 640x480 pixels. The radial distortion is estimated to about $1.4 \cdot 10^{-7}$ proportional to r^3 and $8.5 \cdot 10^{-13}$ proportional with r^5, where r is radius from image center in pixels. These internal camera parameters are estimated off-line using a series of images of the calibration chart.

One image pixel corresponds to an angle of about 0.1° at a 320x240 resolution (The camera opening horizontally is 33.5°). A one-pixel error corresponds to about a 10 cm range error at 2 meter distance from the camera with the used camera configuration. Practical results shows an about 5 cm (standard deviation) distance error from the stereo calculation using a 320x240 image resolution.

A.6 Results

The processing time from image capture to extracted features is shown in Table A.1 for the three available camera resolutions.

The long capture time for 640x480 resolution is due to an interface limita-

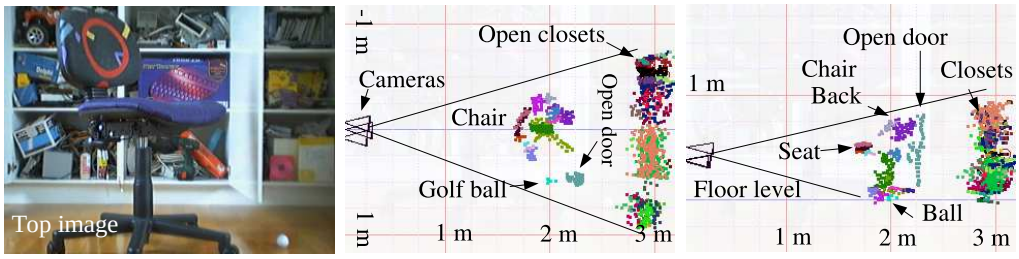


Fig. A.7: The same chair and golf ball as in Figure A.3, but with more complex background. The chair and ball is still extracted. Center image is top view and right image is side view.

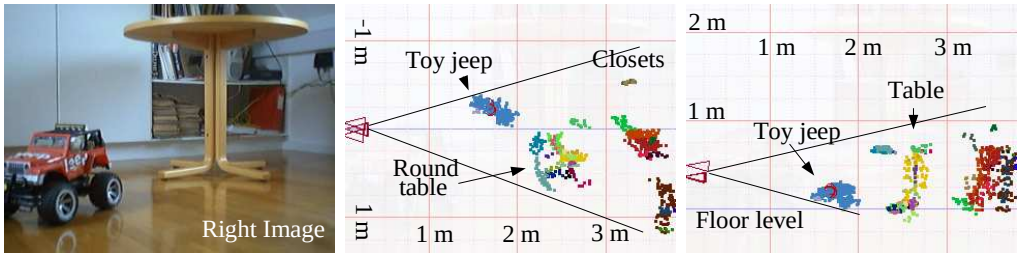


Fig. A.8: The same method now used on a toy jeep and a round table. The objects are extracted to a level where the table should be recognizable as a table.

	160x120	320x240	640x480	
Image delay	0.24	0.24	0.24	sec
Image capture	0.01	0.06	5.8	sec
Epipolar dist.	6	9	12	pix
Stereo scan.	0.53	1.71	4.35	sec
3D detections	487	744	1145	
Feature filt.	0.15	0.11	0.29	sec
Features	19	21	26	
Total time	0.94	2.03	10.68	sec

Table A.1: Vision sensor timing.

tion, where cameras have to be closed between images. All timings are from a 700 MHz PC and for a slightly simpler scene than in Figure A.3.

The main processing time is used for the stereoscopic scanning as would be expected. At the 160x120 resolution, the epipolar distance is six pixels, and as the scanning mask is three pixels wide there is an almost full usage of all pixels

in the image. The result is available after about one second. The 320x240 resolution produces slightly better data, but takes 2 seconds to complete.

In most scenes, the sensor produce from 15 to 40 clusters, where each cluster represents a visible feature from *one* of the objects in the scene.

The delay in camera and interface limits the accuracy of image time stamping, and thereby the accuracy of the position and orientation of the camera, especially if the images was taken during a maneuver. This, in turn, limits the accuracy of the stereo extraction.

The method is rather insensitive to background structure, as can be seen in Figure A.7, where the closed doors have been opened. The method extracts readily more complex shaped objects as the toy Jeep in Figure A.8.

Isolation of features from reasonably sized objects — e.g. the table and the chair — seems adequate for a reasonably safe classification. The 3D extension of objects may be used to get further data from the source images, e.g. additional color or texture information, which can be used to further distinguish objects and object classes. The additional color and texture extraction is not implemented yet; neither is the object classification.

A.7 Conclusion

The vision sensor described in this paper combines stereo vision methods used in e.g. land mapping from aerial photographs, and 3D filtering methods that in total ensures a robust extraction of all — reasonably sized — objects within sight of the robot. The processing time for the purposed method is not prohibitive, and will be even less so in the future.

The extracted objects seem to be adequate for classification of objects in classes like chairs, tables, walls and doors.

The process is designed for an in-door environment, but should also be applicable for out-door use, as there is no dependence on the shape of the seen features.

A.7.1 Next step

The next step is expected to be grouping of detected clusters to isolate non-transparent surfaces, e.g. walls, chair and table surfaces. The image color and texture of these surfaces can then be determined within the surface, to enhance the correlation and classification process.

Appendix B

Navigation script definition

B.1 Introduction

This is a (syntax) description of the implemented control language of the navigation sequencer.

The functionality is controlled by a sequence of statements as defined below:

statements ::= statement [↩ statement]*

statement ::= (assignment_statement | execute_statement | function_definition | flow_control | label | watch_call | print | remark | preprocessor_statement)

↩ ::= '\n' (newline)

assignment_statement ::= variable_name '=' expression

expression ::= value [operator value]

value ::= (expression | '('expression')' | numeric_constant | variable_value | function)

operator ::= ('+' | '-' | '*' | '/' | '>' | '<' | '==' | '!=' | '>=' | '<=' | '&' | '|' | 'and' | 'or' | 'not' | '&&' | '||' | '<<' | '>>')

variable_value ::= [unary_operator] variable_name

unary_operator ::= ('not' | '!' | '-' | '+')

variable_name ::= alpha [(alphanum | '_')]* maximum length 32 characters

function ::= function_name '(' function_params ')'

function_params ::= [expression [, expression]*] number of parameters depend on function.

function_name ::=(library_functions | additional_functions | defined_functions)

library_functions ::= mainly C library functions (see list below).

additional_functions ::= functions written in C (see list below).

defined_functions ::= as defined in script ? see function_definition

numeric_constant ::= ([-] [digit]+['.'[digit]+] ['e'[-] [digit]+]) | ('0x' [digit]+)

B.2 Assignments

An assignment statement could be:

```
dist = hypot(dx, dy) + d*cos(25*pi/180) ? d*sin(25*pi/180)
isok = not ((dist < -2.5) or (dist >= 250))
bit4to8 = (flags \& 0xf8) >> 3
```

The first evaluates a double value and the second a boolean value and the third an integer. All are implemented as double variables and can be mixed. Boolean operators return exact 1.0 (true) or 0.0 (false). Operator precedence is not implemented, except for '*' and '/', that will be evaluated from left to right before any other operators (+, -, and, or, >=, == etc), that will be evaluated from right to left.

B.3 Execute statement

execute_statement ::= exe_identifier exe_params [: stopcondition]

exe_identifier ::= (drive_command | support_command)

drive_command ::= (drive_fwd | drive_turn | drive_gotowaypoint | drive_idle | drive_smrcl)

drive_fwd ::= 'fwd' (('direct' x_dist [, y_dist [, end_h]]) | ('odo' x_dist) | ('left' x_dist) | ('right' x_dist) | ('top' x_dist))

x_dist ::= value (distance (in current or selected direction))

y_dist ::= value (left distance relative to current pose (default is 0.0))

end_h ::= value (end pose heading (default is 0.0))

drive_turn ::= 'turn' angle (angle (value) to turn ? same as 'smrcl turn angle')

drive_gotowaypoint ::= 'gotowaypoint' x_pos y_pos

x_pos ::= value (x coordinate in selected coordinate system).

y_pos ::= value (y coordinate in selected coordinate system).

drive_idle ::= 'idle' idle_time (idle_time (value) is a value in seconds)

drive_smrcl ::= 'smrcl' smrcl_command

smrcl_command ::= (as defined for smrdemo, implicit stop criteria is when smrdemo reports command completed).

support_command ::= (vision | arm | talk | listen | ...) (vision is implemented only)

exe_params ::= depend on exe_identifier.

stopcondition ::= (boolean) expression. (execution continues to next line, when expression is true).

B.3.1 Drive command FWD

The fwd orders the robot to drive forward in one of four methods: left, right, top or direct, e.g.:

`fwd left 20`
Follow left side of road (in `wpfEdgeDist` distance) for 20 m.

`fwd right 30`
Follow right side of road (in `wpfEdgeDist` distance) for 30 m.

`fwd top 50`
Follow top (centre) of road for 50 m. The variable `wpfTopDist` determines the position of the robot relative to the centre line.
If `wpfTopDist` > 0.0, then drive this distance to the left of the centre line. If `wpfTopDist` is negative, then to the right.

`fwd direct 3.5, -1, 1.57`
Drive to the relative position 3.5 m in front of robot, 1 m to the right, and stop in a posture 1.57 (90 deg) counter clockwise relative to current heading. NB! Use comma separated parameters if space separated values could be evaluated (ie '3 -1' is 2 and one parameter only, but '3, -1' is two parameters). Y and H parameters may be omitted, default is 0.0.

`fwd direct gmkX-1.2, gmkY, gmkH-pi :(laserRange(0) < 0.5)`
This command should get the robot to a position 1.2 m in front of a detected guide mark, in a heading facing the guide mark. In this case, the stop criteria tests if the laserscanner gets closer than 0.5 m from an obstacle.

B.3.2 Drive command GOTOWAYPOINT

Drives to a specific coordinate position.

`gotowaypoint odo 652.5, 30.0, 0.0`
This should direct the robot to odometry (x,y) coordinates (652.5, 30.0) and aim for an end heading aligned with the x-axis. The obstacle avoidance route is planned through the best suited corridor for the destination.

`gotowaypoint direct 652.5, 30.0, 0.0`
This directs the robot to odometry (x,y) coordinates (652.5, 30.0) and aim for an end heading aligned with the x-axis. Obstacles are avoided.

`gotowaypoint gps 367986.1, 6235457.4, 0.25` (NOT VALID p.t.)
This should direct the robot towards the GPS coordinates (367986.1 E, 6235457.4 N) and aim for a end heading east (a bit (15 degrees) to the north). NB! This function uses GPS heading, which is badly calculated. If end heading is not specified, then current heading is used.

B.3.3 Drive command IDLE

This command just idles the drive system and waits for a specific number of seconds.

`idle 10`
This command idles the script for 10 seconds.

B.4 Drive command SMRCL

`smrcl 'turnd ' 1 (angle*180/pi ) ' :($odoth >= ' calcH ')'`
This is a SMRCL command for smrdemo, the mmr will try to evaluate all values (that is not in apostrophes), then the expression will be replaced by the evaluated value. Needed white space in the resulting command must be added where needed (except between two evaluated numbers). A syntax error will be the result if parameters can not be evaluated.

The smrcl command shown could be evaluated to: `smrcl turnd 1.0 140.7 :($odoth >= 1.57).` - NB! The explicit stop criteria is evaluated

by smrdemo and not by the mmr, so - the values (here `$odoth`) must be known by smrdemo.

B.5 Function definition

function_definition ::= '`function`' function_identifier '`(`' function_params '`)`' [statements] '`return`' [expression]

function_identifier ::= variable_name

label ::= label_identifier '`:`'

label_identifier ::= variable_name

function_params ::= [variable_name ['`,`' variable_name]*]

B.6 Flow control

flow_control ::= (if_expr | goto_expr | call_expr | skip_drive)

goto_expr ::= '`goto`' label_identifier

call_expr ::= ['`call`'] defined_function

defined_function ::= function_identifier '`(`' [expression ['`,`' expression]*] '`)`'
if_expr ::= '`if`' '`(`' (boolean) expression '`)`' then_expr

then_expr ::= flow_control | assignment_statement

Flow control statements could be:

```
if (not defined(doorNum)) doorNum = 0
if (doorNum < 7) goto end\_label
call waitAWhile(10)
```

An if statement has one conditional statement only, so a '`goto`' or '`call`' construction must be used if more statements are to be conditional.

B.7 Skip statement

A skip statement is an extended stop-condition for an executable statement. Skip statements can therefore only be used in watch-functions. It will interrupt the main program flow, as either a jump or a function call. The watch function flow continues unaffected.

skip_drive ::= (skip_to_label | skip_function_call)

skip_to_label ::= '`skip`' [label_identifier] (if no label, then skip to next line)

skip_to_function ::= '`skipcall`' defined_function

In a watch function a skip line could be:

```
if (doorNum > 25) skip to_end
```

watch_call ::= 'watch' watch_name ['every' value] (call_expr | assignment_statement | print) 'unwatch' watch_name

Watch statements starts evaluation of a watch function (or statement) at regular intervals (as defined by the 'every' art) or at sample rate (p.t. 3Hz), until an 'unwatch' statement is executed. Examples could be:

```
watch max_speed maxSpeed = max(speed, maxSpeed)
watch look4door()
```

The first executes an assignment statement at every sample time, the second calls a '`look4door()`' function, and uses the short form, where the watch name is the same as the function call.

print ::='`print`' [print_text | expression]+

print_text ::= '’' text '’' | '"' text '"'

Print statements are simple statements, that sends a string and a value to the client. The expression evaluates to a double. An example could be:

```
print 'Found door number ' doorNum ' but expected ' doorCnt
print 'Finished'
```

this text and the evaluated values will be send to all connected clients as an XML tag, eg:

```
<help info="Found door number 6 but expected 5"/>\n>.
```

B.8 Remarks

Remark ::= ('%' | '//' | ';') [any_non_control_character]*

The rest of the line is disregarded in the statement evaluation.

B.9 Library functions

The following standard math library functions are pt implemented: `sin(a)`, `cos(a)`, `asin(a)`, `acos(a)`, `tan(a)`, `atan(a)`, `atan2(y, x)`, `hypot(a, b)`, `sqrt(a)`, `sqr(a)`, `abs(a)`, `min(a, b)`, `max(a, b)`.

B.10 Special functions

The following robot or navigation specific functions are pt implemented.

`limitToPi()`
: Returns a value in the [-pi..pi] range.

`defined(varname)`
: Returns true if 'varname' is defined.

`calcPoseAtTime(time)`
: Returns posture (in calcX -Y -H) at time 'time'.

`calcPoseAtDist(near)`
: Returns the time and the posture at distance near from current position.

`calcPoseFitAtDist(near, far)`
: Calculates a line fitting of the positions in this interval and returns the result as posture (x,y,h) on line close to the 'near' position.

`calcAddRel(x, y)`
: Adds this offset relative to the calculated posture in calcX -Y -H.

`distToOdoEnd()`
: Returns the distance from current position to odometry destination end position.

`distTo(odox, odoy)`
: Returns the distance to the specified position.

`distToGps(utmE, utmN)`
: Returns the distance to a GPS (UTM) position.

`distToPoseLine(pose_x, pose_y, pose_h, x, y)`
: Returns distance from (x,y) position to the posture line defined by pose_x, _y, _h.

`distToPoseLineSigned(pose_x, pose_y, pose_h, x, y)`
: Returns the distance from road direction line signed, left is positive.

`headingGps(dist)`
Returns current heading 'u' (radians) based on odometry and GPS(UTM) position now and at 'dist' back from current position. (East is zero CCV, to get compas radians use: (pi/2 - u).

`laserRange(angle)`
Returns laser range measurement closest to this angle (angle 0.0 deg is font).

B.10.1 Guidemark functions

Guidemark functions work on data returned by camera server following a '`gmkget`' command to the camera server, e.g. by statement: '`vision gmkget extra=true`'.

The functions are:

`gmkCnt()` returns count of guidemark positions available.

`gmkNewCnt(t)` returns number of stored guidemarks since 't'. The 'time' is stored and is used by the 'gmkNewNum(N)' call.

`gmkNewNum(t, N)` returns index to guidemark N of the new guidemarks

`gmkNum(idx)` returns true if guidemark with index idx is found, and sets the guidemark values (from 'cam getgmk') in variables: gmkX, gmkY, gmkZ, x, y and z position gmkO, gmkP, gmkK, rotation on x,y and z axis gmkTime, gmkID detection time and guidemark ID code.

`gmkNewID(t, ID)` returns true if a guide mark newer than t is available with this ID, the guidemark values are set as for the 'gmkNum(idx)' call.

`gmkID(ID)` returns true if a guide mark is stored with this ID, and the guidemark values are set as for the 'gmkNum(idx)' call.

B.11 System defined variables

The following variables are available in the system. The values shown are mostly just examples. The values can be queried or changed from a script or from a client.

The values are formatted under the following headings:
`(index)` `value` `'variable name'` remark.

Constants

```
#000          0.0000 'false'.
#001          1.0000 'true'
#002          3.1416 'pi'.
```

Road data

```
#003          0.0000 'roadValid' is road (width) data valid.
#004          0.7812 'roadWidth' estimated road width [m].
#005          0.0000 'roadQual' road width quality [0..1].
#006          1.0000 'roadTypeLaser' estimated road type.
#007          0.0044 'roadRoughness' current road roughness.
#008          0.0000 'roadLeftValid' left side of road valid.
#009        100.0000 'roadLeftDist' from robot to left side [m].
#010          0.0000 'roadLeftQual' road side quality [0..1].
#011          0.0000 'roadRightValid' right side of road valid.
#012        100.0000 'roadRightDist'  from robot to right side [m].
#013          0.0000 'roadRightQual' road side quality [0..1].
#014          0.0000 'roadTopValid' top of road valid.
#015        100.0000 'roadTopDist' to top-line from robot left is pos.
#016          0.0000 'roadTopQual' road top quality [0..1].
#017          2.3416 'roadCenterX' road centre (top) odometry X [m].
#018         -0.7940 'roadCenterY' road centre (top) odometry Y [m].
#019          0.0863 'roadCenterZ' road centre (top) odometry Z [m].
```

Current pose

```
#020          0.0000 'posex' odometry pose X [m].
#021          0.0000 'posey' odometry pose Y [m].
#022          0.0000 'poseh' odometry pose Z [m].
#023          0.0000 'odoSpeed' current robot velocity [m/s].
#024 1141403388.6506 'time' odometry pose update time [sec].
#025          0.0000 'distOdo' total odometry based distance driven.
```

GPS position (in UTM for current zone)

```
#026          0.0000 'ekfE' Easting.
#027          0.0000 'ekfN' Northing.
#028          0.0000 'ekfh' Heading, p.t. not reliable (East = 0.0, rad).
#029 1141403388.6506 'ekftime' latest GPS update time [sec].
```

Execute statement status

#030 0.0000 'startPosex' pose at start of current drive command.
#031 0.0000 'startPosey'.
#032 0.0000 'startPoseh'.
#033 0.0000 'startTime'.

Drive command stop condition variables

Most variables are only updated when used in a stop-condition in the current (drive) command.

#034 0.0000 'distSoFar' distance since start of drive command.
#035 0.0000 'distTotal' distance since start of server.
#036 0.0000 'endPoseX' current destination position X.
#037 0.0000 'endPoseY'.
#038 0.0000 'endPoseH'.
#039 0.0000 'endPoseDist' Distance remaining to destination.
#040 1141403388.6506 'endPoseTime'.
#041 0.0000 'endPoseSpeed'.
#042 0.0000 'endEkfE'.
#043 0.0000 'endEkfN'.
#044 0.0000 'endEkfH'.
#045 0.0000 'endEkfDist'.
#046 0.0000 'timeout'.

Drive command parameters

#047 0.7000 'speed' current target speed (max).
#048 0.4000 'acc' allowed acc (lateral) [m/s2].
#049 0.4500 'accTurn' allowed centripetal acc [m/s2].
#050 0.0000 'driveMode' 0=idle, 1=smrcl, 2=Laser/vision.
#051 0.0000 'driveState' 0=idle, 1=smrcl, 2=Laser/vision.
#052 0.7100 'wpfEdgeDist' Dist. to edge of road, when 'fwd left' or 'right'.
#053 0.0000 'wpfTopDist' Dist to top of road, when 'drive top' pos=left.

Result values of pose calculations

#054 0.0000 'calcX'
#055 0.0000 'calcY'

```
#056          0.0000 'calcH'
```

Historic heading

Historic heading based on the last 5 m, excluding the last 1 m.

```
#057          0.0000 'histH' [radians]
#058       3600.0000 'histHAge' age of estimate (3600 is invalid) [sec]
#059          1.0000 'histW' Weight of hist heading, when selection best
route
```

Used angle of laser scanner relative to horizontal

```
#060         -0.1683 'laserTilt' positive is UP [radians]
```

Parameters for laser scanner based road detection

These variables are mostly for debugging and parameter experiments.

```
#061          0.1000 'wpfDevLimit'
#062          0.0075 'wpfConvex'
#063          0.0250 'wpfDevOffset'
#064          4.5000 'wpfEndDev'
#065          1.0000 'wpfSmooth'
#066          1.0000 'wpfSmoothAuto' uses roadTypeLaser
#067          0.0000 'wpfMode'
#068          0.0000 'wpfIgnoreObstacles' debug flag
#069          1.0000 'wpfUseIntervalCombiner'
#070          1.0000 'wpfOutdoorObstacles'
```

Obstacle maintenance and use

```
#071          2.0000 'obstUseCnt' obstacle groups in obstacle avoid.
#072          7.0000 'obstGrpDist' obstacle grouping max distance.
#073          7.0000 'obstGrpTime' obstacle grouping max time.
```

Guide mark data for selected guide mark

```
#074          0.0000 'gmkX'
#075          0.0000 'gmkY'
#076          0.0000 'gmkZ'
#077          0.0000 'gmkO' - rotation on x-axis (roll) positive clockwise
#078          0.0000 'gmkP' - rotation on y-axis (pitch) down is positive
```

#079 0.0000 'gmkK' - rotation on z-axis (yaw) left is positive, zero is front
#080 0.0000 'gmkTime' - guide mark update time
#081 0.0000 'gmkId' - ID code in guide mark (decimal)

Vision items

```
#082          1.0000 'visPathUse'
#083          0.4300 'visPathBal'
#084          0.6500 'visPathLim'
#085          1.0000 'visPathInterval'
#086          0.0000 'visCamTiltErr'
#087          0.0030 'visCamTiltGain'
```

Simulation

Simulation flag and time, used during replay of logdata.

```
#088          0.0000 'simulated'
#089 1141403388.6689 'simTime'
```

B.12 Examples

B.12.1 A 4 m square

A 4 m square script.

```
// function for one leg ? drive to position and turn
function drive\_and\_turn(x, y, ang)
gotowaypoint odo, x, y
smrcl turn ang
return
//
// drive on odometry (not gyro) (optional)
odo = 0
gyro = 1
smrcl 'set odocontrol' odo
// set drive speed to be used
speed = 0.6
// set angle and distance variables
angle = 90
d = 4
if (angle < 0) y = -d
```

```
drive\_and\_turn(d, 0, angle)
drive\_and\_turn(d, d, angle)
drive\_and\_turn(0, d, angle)
drive\_and\_turn(0, 0, angle)
// stop
smrcl idle
```

B.12.2 Up and down the hallway

A script that follows the walls up and down a corridor.

```
// follow left wall and stop where started
speed = 0.5
// drive distance
dist = 25
fwd left dist
// turn right to avoid wall
smrcl turn -175
// go back
fwd right dist
smrcl turn 179
// stop
smrcl idle
```

B.12.3 Up and down the hallway with stops

The same as the previous script, but extended to check for open doors and makes a pause at every open door to the right

```
//# use watch function to look for doors
//# use gyro for heading
smrcl 'set "odocontrol" 1'
smrcl 'set "$gyro1off" 0.32'
speed=1
door = true
doorNow = false
function look_for_door(direc)
  ld = laserRange(direc)
  doorNow = (ld > 2.5)
  if (not door and doorNow and (distSoFar > 1))
                          skipcall doWait(5)
  if (door != doorNow) print "laser distance " ld
  door = doorNow
```

```
  # print "At open door number " door
return door
//#
function doWait(waitTime)
  dd = dd + distSoFar
  print "Dist so far is now " dd " m"
  smrcl 'fwd ' 0.3 ' @a 0.3'
  smrcl 'idle'
  idle waitTime
return
//#
//# make variables for left and right direction
left = 90
right = -90
//# start watch for doors to the right
watch doors look_for_door(right)
//#
//# start drive
dist = 25
dd = 0
fwd left dist :((dd + distSoFar) > dist)
//# stop watch before turning
unwatch doors
//# turn right to avoid wall
smrcl 'turn -175'
//# go back
dd = 0
watch doors look_for_door(left)
fwd right dist :((dd + distSoFar) > dist)
unwatch doors
smrcl 'turn 179'
//# stop
smrcl 'idle'
```

B.12.4 Test area navigation script

```
//# Dyrehaven full route
//# run on gyro (1), (odo is 0)
smrcl 'set "odocontrol" 1'
//# gyro offset 0.445 19 aug, 0.32 24 aug,
//#              0.19 12 sep, 0.25 17 okt 2005
smrcl 'set "$gyro1off" 0.25'
//#
//# shortcut jump if about 50 m before 5-vej
// goto lbl5Vej
// goto bagslot
///////////////// Hjortekaervej ////////////////////////
//# get going from anywhere on hjortekaervej - if GPS works
acc = 0.4
accturn=0.9
vispathlim=0.65
vispathbal=0.43
vispathuse=true
//# start slow
wpfedgedist = 0.85
speed = 0.7
fwd left 5
//# high speed to just before castle (should be ~1660 m)
//# stop ~20m before end of road
print "To cruse speed"
speed = 1.4
dist = 1740
fwd left dist :(25 > distToGps(347716.7, 6183490.1))
//
////////////// Royal hunting lodge square ////////////////
print "Close to Royal hunting lodge"
//# change to gyro heading for crossing
//       of gravelled square (and rest of the way)
smrcl 'set "odocontrol" 1'
//# reduce speed - close to open area
speed = 0.9
//# calculate direction of hjortekrvej
calcPoseFitAtDist(0, 25)
//# add to a waypoint inside castle square
speed = 0.85
//# follow left side into square -
```

```
//# until fair distance to hjortekrvej line
fwd left 75 :(7.0 < abs(distToPoseLineSigned(
                      poseX, poseY, calcX, calcY, calcH)))
//# get better reference close to end of hjrtekrvej
calcPoseFitAtDist(12, 25)
//# add to a point far away in Ndr. Eremitagevej direction
//# - relative to hjortekrvej ref point (GPS-based)
calcAddRel(75.0, 150.0)
//# go towards that point - first just avoid obstacles
//# gotowaypoint odo calcx calcy :(distsofar > 10)
//# then look for road edge
gotowaypoint odo calcx calcy :((roadLeftQual > 0.78) and
                (roadLeftDist < 2.7) and (distsofar > 10))
//
///////////////// Behind lodge ////////////////////////
//# assume left edge is found - continue close to left edge
wpfEdgeDist = 0.85
fwd left 40
//# now we are on asphalt, so get closer to edge and speed up
wpfEdgeDist = 0.65
speed = 1.3
bagslot:
//# continue until just before end of asphalt
fwd left 290 :(30 > distToGps(348053.0, 6183676.0))
print "close to end of asphalt - going top step 1 (for 5m)"
//# switch to top of road (in 2 steps to be smooth)
td = roadTopDist / 2
wpfTopDist = td
fwd top 5
print "close to end of asphalt - going top step 2 (280m)"
//# drive on road centerline - until (25) before junction
lbl5Vej:
wpfTopDist = 0
speed = 1.0
fwd top 280 :(25 > distToGps(348223.0, 6183901.0))
//
//////////////////// 5 vej ////////////////////////////
//# reduce speed
speed = 0.8
calcPoseFitAtDist(0, 20);
//# Follow road to the left, up Chausseen (north)
wpfEdgeDist = 1.1
```

```
fwd left 100 :(7.0 < abs(distToPoseLineSigned(
                    poseX, poseY, calcX, calcY, calcH)))
print "Found Chausseen - at distSoFar=" distSoFar
//# turn to springforbi road
turn -58
smrcl 'fwd 2.5 @v0.8'
//# head for 15 more meter in direction of Eremitvej
frX = poseX - calcX
frY = poseY - calcY
frd = hypot(frX, frY)
calcAddRel(frd + 20, 0)
//# goto entry of right road
gotowaypoint odo calcx calcy :(endPoseDist < 7)
//# advance waypoint up the road, and
//#         go that way until top is found
calcAddRel(100, 0)
gotowaypoint odo calcx calcy :(roadTopQual > 0.85)
print "Found top of road - Q=" roadTopQual
//# follow top until just before horsetrack
dist = 7
fwd top dist
calcposefitatdist(0, 6)
calcaddrel(100,0)
//# go direction to waypoint until passed the horsetrack
gotowaypoint odo calcx calcy :(distSoFar > 12)
//
///////////////// Springforbi ////////////////////////////
//# increase to crouse speed and continue to goal
speed = 0.9
fwd top 350 :(30 > distToGps(348485.0, 6184198.0))
print "Found the end - distTotal=" distTotal
//
//# stop
speed = 0.0
smrcl fwd 0.5 @v0.3 @a0.3
smrcl idle
print "The end."
//# fin
```

www.oersted.dtu.dk

Ørsted•DTU
Automation
Technical University of Denmark
Building 348
DK-2800 Kgs. Lyngby
Denmark
Tel: (+45) 45 25 38 00
Fax: (+45) 45 93 16 34
E-mail: info@oersted.dtu.dk

ISBN 87-91184-64-9